Transformations d'Arbres XML avec des Modèles Probabilistes pour l'Annotation

Florent Jousse

Transformations d'Arbres XML avec des Modèles Probabilistes pour l'Annotation

THÈSE

présentée et soutenue publiquement le 31 Octobre 2007

pour l'obtention du

Doctorat de l'Université Charles de Gaulle - Lille 3

(spécialité informatique)

par

Florent Jousse

Composition du jury

Rapporteurs :	Stéphane Canu, Professeur	INSA de Rouen
	François Yvon, Professeur	Université Paris Sud
Examinateurs :	François Denis, Professeur	Université de Provence
	Isabelle Tellier, Maître de Conférence	Université Charles de Gaulle - Lille 3
	Marc Tommasi, Maître de Conférence	Université Charles de Gaulle - Lille 3
Directeur :	Rémi Gilleron, Professeur	Université Charles de Gaulle - Lille 3

UNIVERSITÉ LILLE 3
Groupe de Recherche sur l'APPrentissage Automatique

Mis en page et composé avec LaTeX

Ce que nous devons apprendre à faire,
nous l'apprenons en le faisant.
Aristote

Human beings, who are almost unique in having
the ability to learn from the experience of others,
are also remarkable for their apparent disinclination to do so.
Douglas Adams

Remerciements

Je tiens tout d'abord à remercier Rémi Gilleron, directeur de cette thèse, ainsi qu'Isabelle Tellier et Marc Tommasi qui ont co-encadré mes travaux. La confiance et la liberté qu'il m'ont accordées, ainsi que les réunions, parfois très animées, que nous avons pu avoir tout au long de cette thèse ont permis à ces trois années de se dérouler dans les meilleures conditions possibles.

Je remercie aussi l'ensemble de l'équipe GRAppA de l'université de Lille 3, ainsi que tous les membres de l'équipe MOSTRARE. Merci tout particulièrement à Hahn-Missi Tran qui s'est chargé avec brio du développement du package XCRF, et qui a supporté mes incessantes demandes de modifications. Merci aussi à Daniel Muschick avec qui j'ai eu l'occasion de travailler dans le cadre de son Master 2. Son enthousiasme et sa motivation ont eu un impact positif sur mes travaux. Enfin, un grand merci à toute l'équipe et à mes camarades de bureau qui ont contribué à faire reigner une ambiance à la fois studieuse et agréable : les vendredi après-midi de la "petite maison" et du "loft" me manqueront.

De plus, je remercie tout particulièrement Stéphane Canu et François Yvon qui ont accepté de rapporter cette thèse, ainsi que François Denis qui me fait l'honneur de présider le jury.

Enfin, je remercie mes amis et ma famille qui m'ont soutenu et ont supporté mes humeurs durant ces derniers mois de rédaction. Un grand merci notamment à ma sœur Marina qui, en plus de m'apporter son soutien moral, a donné de son temps pour relire cette thèse.

Résumé

Cette thèse traite de l'apprentissage de transformations d'arbres XML à l'aide de modèles probabilistes pour l'annotation. Pour cela, nous adaptons dans un premier temps au cas des arbres le modèle des champs aléatoires conditionnels (CRF), souvent utilisé pour l'apprentissage d'annotations de séquences. Cette adaptation porte à la fois sur le modèle de dépendances et sur les algorithmes d'inférence exacte et d'apprentissage. Nous apportons aussi deux méthodes d'amélioration de la complexité de ces algorithmes afin de permettre leur utilisation dans le cadre d'applications à grande échelle. Ces méthodes s'appuient toutes deux sur les connaissances du domaine pour améliorer la complexité, et consistent d'un côté en l'intégration de contraintes sur l'annotation, et de l'autre en l'approximation d'un CRF par plusieurs CRFs de complexité moindre.

Table des matières

Table des figures

Liste des tableaux

Introduction

1 Domaine et motivations

Dans l'évolution récente du Web, les données semi-structurées jouent un rôle prépondérant. L'exemple principal en est le langage XML (*eXtensible Markup Language*). En effet, celui-ci permet de décrire des données sous forme d'arbres, dont la structure est définie par un schéma. Il est par conséquent devenu le standard en termes d'échanges de données, que ce soit sur le Web ou entre plusieurs applications. Toutefois, les documents XML peuvent avoir des structures très variables. En effet, ceux-ci peuvent être soit orientés données, c'est-à-dire avoir une structure qui apporte une sémantique au contenu textuel, à la manière du schéma d'une base de données relationnelle, soit orientés documents et ainsi avoir une structure dont le but est plutôt de décrire l'organisation ou la présentation. C'est par exemple le cas des pages Web au format XHTML. De plus, parmi ces documents XHTML, la présentation, et donc la structure arborescente, est une fois de plus très variable.

La grande variété des structures des arbres XML nécessite d'être capable de transformer de tels arbres. D'une part, afin de permettre communication entre plusieurs applications, il est nécessaire que celles qui reçoivent les données soient capables de les comprendre. Elles doivent pour cela connaître le schéma XML de ces arbres. Ainsi, il est nécessaire de transformer les arbres XML envoyés par une application dans ce nouveau schéma connu. L'autre nécessité provient de la réutilisation des données du Web. En effet, le Web est rapidement devenu la plus grande source d'informations au format XML, Google évaluant en 2005 le volume de données du Web à 5 millions de téraoctets. La nécessité de réutiliser ces données au sein d'applications se fait donc de plus en plus forte. On peut par exemple imaginer des applications de veille sur le Web, permettant de se tenir informé des évolutions et nouveautés de domaines aussi variés que l'apprentissage automatique, le réchauffement climatique ou les baladeurs MP3. De telles applications doivent alors être capables d'interpréter et réutiliser les informations contenues sur les sites traitant de ces domaines. Pour pouvoir réutiliser ces données, il est donc nécessaire que les applications les reçoivent dans un format XML qu'elles connaissent et par conséquent savent interroger, ce qui implique une fois de plus la transformation des documents XML dans un format connu. Dans le cas de l'application de veille sur des sites Web, un schéma possible est celui des flux RSS, qui est un format XML de syndication permettant de lister des articles ayant chacun des informations telles qu'un titre, une description ou encore un lien vers la source de cette article.

Pour effectuer des transformations d'arbres XML, deux possibilités s'offrent à nous.

Il est dans un premier temps possible d'écrire, pour chaque source d'informations, des programmes de transformation d'arbres XML. Pour cela, le W3C propose par exemple les langages de transformation XSLT (*eXtensible Stylesheet Language Transformations*) et XSLT 2.0[1]. D'autre langages permettant la manipulation de documents XML tels que Python ou Perl peuvent aussi être utilisés pour écrire des scripts de transformation d'arbres XML. Écrire de tels programmes reste néanmoins coûteux en temps et nécessite de l'utilisateur la connaissance de langages de programmation et de transformations. De plus, toute modification de la structure de la source d'informations impose l'écriture d'un nouveau programme. Pour éviter ces problèmes, la solution consiste alors à utiliser des techniques d'*apprentissage automatique supervisé*, c'est-à-dire à implémenter des méthodes permettant d'apprendre, à partir d'exemples, des programmes capables de réaliser une certaine tâche. Dans notre cas, ces techniques ont pour but d'apprendre automatiquement des programmes chargés de réaliser les transformations d'arbres XML à partir d'exemples de transformations d'arbres, c'est-à-dire de documents XML d'entrée et de sortie. Avec de tels systèmes, l'utilisateur n'a alors plus besoin de connaître les langages de transformation d'arbres XML, son rôle se limitant à fournir au système ces exemples de transformations. Dans le cas de l'application de veille de sites Web, un utilisateur devra par exemple annoter sur quelques pages Web les informations qui l'intéressent comme le titre, l'auteur ou encore le résumé de chaque article, pour permettre au système d'apprendre un programme capable de les identifier correctement et de réaliser ces transformations. De plus, avec un système d'apprentissage automatique, si la structure d'une source d'information change, l'utilisateur doit uniquement fournir de nouveaux exemples au système afin d'apprendre un nouveau programme de transformation d'arbres XML.

Dans le cadre des transformations les plus simples, c'est-à-dire dans le cas où la sortie ne possède pas de structure arborescente, des solutions ont été apportées. En effet, plusieurs systèmes permettent d'apprendre automatiquement des programmes d'extraction d'informations, communément appelés *wrappers*, capables d'identifier diverses informations, telles que des titres, noms d'auteur ou encore des dates, dans des documents semi-structurés tels que des arbres XML. Ces informations peuvent être extraites séparément dans le cadre de l'extraction d'informations monadiques, ou sous la forme de tuples (ou relations), par exemple (titre,auteur,date), dans le cadre de l'extraction n-aire. Dans le premier cas, les travaux de [Kosala et al., 2003, Cohen et al., 2003, Carme et al., 2004a], basés sur des automates d'arbres, furent parmi les premiers à utiliser la structure arborescente des arbres XML pour effectuer des tâches d'extraction d'informations monadiques avec succès. Dans le cas de l'extraction n-aire, les travaux de [Gilleron et al., 2006], utilisant des méthodes de classification multi-classes, ou encore ceux de [Muslea et al., 2001, Lemay et al., 2006] ont aussi permis de résoudre ces tâches en tenant compte de la structure arborescente des documents XML.

L'apprentissage automatique ayant déjà fait preuve de bonnes performances pour ces transformations simples, nous considérons donc dans cette thèse l'apprentissage de transformations d'arbres XML plus difficiles. En effet, nous souhaitons apprendre à effectuer des transformations où les données de sorties possèdent une véritable structure arborescente. Nous nous limitons toutefois à une classe de transformations plus simples : celles

[1] `http://www.w3.org/TR/xslt20`

pouvant être modélisées par une annotation de l'arbre d'entrée. Pour cela, nous utilisons des modèles probabilistes pour l'annotation, plus précisément les champs aléatoires conditionnels, que nous adaptons au cas des arbres XML.

2 Contribution

Cette thèse traite donc de l'apprentissage de transformations d'arbres XML. Nous y montrons que l'utilisation de modèles probabilistes pour l'annotation est un bon choix pour apprendre à transformer des arbres XML en tenant compte de leurs spécificités.

Nous choisissons ici de nous limiter aux transformations pouvant être modélisées par une annotation de l'arbre d'entrée, c'est-à-dire en affectant à chaque nœud de l'arbre d'entrée un label définissant comment effectuer la transformation. Pour cela, nous proposons dans cette thèse deux méthodes d'annotation d'arbres XML permettant d'effectuer des transformations. La première nécessite la connaissance préalable du schéma cible et consiste en l'annotation des feuilles textes de l'arbre d'entrée avec leur position dans l'arbre de sortie. La seconde méthode, quant à elle, consiste à annoter les nœuds des arbres XML d'entrée avec des opérations d'édition d'arbres du type : renommer ou supprimer un nœud, supprimer un sous-arbre, *etc.* Les transformations d'arbres ainsi représentées sont relativement simples et ne permettent par exemple pas de changer l'ordre des fils d'un nœud. Toutefois, elles permettent une expressivité suffisante pour de très nombreuses tâches de transformation d'arbres XML.

Pour effectuer ces annotations d'arbres XML, nous nous sommes tournés vers les modèles probabilistes, et plus précisément la famille des modèles graphiques, qui permettent de représenter une distribution de probabilité. Parmi les modèles graphiques, nous avons choisi d'opter pour des modèles conditionnels, ceux-ci représentant la probabilité conditionnelle d'une annotation sachant l'arbre XML d'entrée (l'observation). Un tel modèle fournit alors un système d'annotation d'arbres, dans lequel l'annotation choisie est celle maximisant cette probabilité conditionnelle, pour un arbre XML d'entrée donné. Le choix de ces modèles conditionnels a principalement été guidé par le fait que nous voulions un modèle robuste aux variations. En effet, dans les tâches de transformations que nous abordons, les arbres XML d'entrée sont sujets à de nombreuses variations, que ce soit au niveau de leur structure interne ou de leur contenu textuel. Ainsi, lors de l'annotation, il est très fréquent de rencontrer des configurations qui n'apparaissent pas parmi les exemples utilisés lors de l'apprentissage. L'utilisation de modèles graphiques permet une grande robustesse à ces variations.

De plus, une des difficultés des tâches de transformations d'arbres XML réside dans la nécessité d'utiliser à la fois la structure interne et le contenu textuel des documents XML pour guider les transformations. Dans ce but, notre choix s'est tourné vers le modèle des champs aléatoires conditionnels [Lafferty et al., 2001] ou *Conditional Random Fields* (CRFs). En effet, ceux-ci permettent de calculer la probabilité conditionnelle de l'annotation sachant l'observation en utilisant des fonctions de caractéristiques. Ces fonctions, la plupart du temps à valeur binaires, voient leur valeur conditionnée par des tests sur l'ensemble de l'observation. Ceux-ci peuvent ainsi porter à la fois sur la structure et sur le contenu textuel. Toutefois, le modèle des CRFs n'avait jusqu'à présent été que très peu

adapté à l'annotation d'arbres, à l'exception des travaux de [Cohn and Blunsom, 2005] dans le domaine du traitement des langues naturelles, et jamais adapté au cas des arbres XML. Nous proposons donc dans cette thèse *l'adaptation des champs aléatoires conditionnels au cas de l'annotation d'arbres* XML. Nous définissons dans ce but trois modèles de CRFs de complexités croissantes. De plus, nous comparons le modèle le plus expressif, appelé 3-CRF, aux automates d'arbres descendants stochastiques et montrons que, dans le cas des arbres binaires, les distributions de probabilité conditionnelle définies par ces automates peuvent être définies par des 3-CRFs.

Une autre difficulté liée aux arbres XML est le problème de la complexité des algorithmes d'apprentissage et de recherche de la meilleure annotation. En effet, deux caractéristiques des arbres XML sont à gérer avec précaution. D'une part, la taille des arbres XML peut être parfois très grande, dépassant ainsi les 5000 nœuds pour un arbre. De plus, les arbres XML sont des arbres d'arité non bornée, c'est-à-dire qu'il n'y a pas de limite sur le nombre de fils d'un nœud. Pour tenir compte de ces caractéristiques, nous avons *adapté les algorithmes d'inférence exacte et d'apprentissage* utilisés dans les CRFs sur les séquences au cas des arbres XML. Cette adaptation permet de passer outre les problèmes de taille et d'arité des arbres XML. En effet, ces algorithmes, s'inspirant à la fois des algorithmes sur les séquences (Viterbi, *Forward-Backward*) et de l'algorithme *Inside-Outside* des grammaires hors-contextes probabilistes, sont linéaires dans la taille des arbres.

Toutefois, ces algorithmes possèdent un goulot d'étranglement. En effet, leur complexité est, dans le cas des 3-CRFs, cubique dans le nombre de labels de la tâche d'annotation. Pour remédier à cette complexité et ainsi permettre l'utilisation des 3-CRFs dans le cadre de tâches de transformation à grande échelle, nous proposons dans cette thèse deux méthodes permettant de *réduire la complexité des algorithmes* d'inférence et d'apprentissage tout en *utilisant les connaissances du domaine*. Ces deux méthodes, bien que présentées ici dans le cadre des CRFs pour les arbres XML, sont utilisables avec tous types de CRFs. La première technique que nous proposons consiste à *intégrer des contraintes* dans les champs aléatoires conditionnels. Ces contraintes viennent restreindre l'espace des annotations possibles d'un arbre en interdisant des configurations de labels au niveau des cliques. La connaissance préalable de ces restrictions peut venir, par exemple, de la connaissance du schéma des arbres XML de sortie, ou encore de la sémantique associée aux différents labels. Nous adaptons alors les algorithmes d'inférence exacte et d'apprentissage des champs aléatoires conditionnels afin de leur faire prendre en compte les contraintes ainsi définies. Enfin, nous montrons par une série d'expériences que ces contraintes permettent non seulement d'améliorer la complexité des algorithmes, mais entraînent aussi une hausse des résultats de par la possibilité qu'elles offrent d'utiliser les connaissances du domaine.

La seconde technique d'amélioration de la complexité que nous proposons consiste à composer plusieurs CRFs définis sur des sous-parties de l'alphabet des labels pour approximer un CRF défini sur l'ensemble de cet alphabet. Dans cette thèse, nous décrivons *trois méthodes de composition de* CRFs. Les deux premières s'appuient sur la connaissance préalable d'une partition de l'alphabet des labels. Tandis que l'une consiste à effectuer les différentes sous-tâches d'annotation en parallèle, l'autre méthode de composition suppose l'existence d'un ordre total sur la partition de l'alphabet des labels. De ce fait, les différentes tâches d'annotation sont effectuées séquentiellement, chacune bénéficiant des

résultats des annotations précédentes. Enfin, la troisième méthode considère pour sa part l'existence d'une hiérarchie sur l'alphabet des labels. Celle-ci permet alors de diviser l'alphabet des labels en plusieurs sous-parties. Les différentes sous-tâches d'annotation ainsi définies sont alors effectuées en combinant les méthodes de compositions parallèle et séquentielle. En divisant une tâche d'annotation sur un alphabet de labels de grande taille en plusieurs sous-tâches ayant un nombre de labels beaucoup plus faible, ces méthodes de compositions permettent de réduire grandement la complexité des algorithmes d'inférence et d'apprentissage. Nous montrons aussi expérimentalement que, en plus d'une réduction considérable de la complexité, ces méthodes de composition permettent d'obtenir des résultats comparables à ceux obtenus avec des 3-CRFs définis sur l'ensemble de l'alphabet des labels.

Enfin, la dernière contribution de cette thèse réside dans la mise en pratique de ces concepts dans le cadre d'une application de génération automatique de flux RSS à partir de pages Web. Cette application permet à son utilisateur d'apprendre un générateur de flux RSS en annotant une ou plusieurs pages d'un site Web. Ce générateur consiste en un champ aléatoire conditionnel qui annote les pages Web de ce site de façon à transformer l'arbre XHTML en un arbre XML au format RSS. Il permet alors de créer automatiquement des flux RSS pour toutes les pages du même site Web. Le bon fonctionnement de cette application met en lumière la qualité de notre adaptation des champs aléatoires conditionnels au cas des arbres XML, ainsi que celle des techniques d'amélioration de la complexité des algorithmes dans la but de l'application sur des données réelles.

3 Travaux associés

Bien que les transformations d'arbres XML soient un domaine de recherche récent, on trouve plusieurs autres travaux sur le sujet. Nous choisissons de positionner nos travaux par rapport à cinq d'entre eux. Ceux-ci ne représentent naturellement pas une liste exhaustive. Dans un premier temps, les travaux de [Chidlovskii and Fuselier, 2005] traitent du problème de la transformation d'arbres HTML en arbres XML dont le schéma est connu. Pour cela, ils réduisent eux aussi ce problème de transformation à une tâche d'annotation des arbres HTML d'entrée. Ainsi, les documents HTML sont annotés sémantiquement par leur correspondant dans le schéma XML cible. Dans une première étape, les feuilles des arbres XML de sortie sont prédites à l'aide d'un classifieur à maximum d'entropie. La structure de la sortie est ensuite déterminée au moyen d'une grammaire hors-contexte probabiliste ou PCFG (*Probabilistic Context-Free Grammar*). Cette technique fait toutefois l'hypothèse que les feuilles sont dans le même ordre dans les arbres d'entrée et de sortie. De plus, la complexité de la deuxième étape est cubique dans le nombre de feuilles. Ainsi, ce système ne peut être appliqué aux tâches que nous considérons ici dans lesquelles les arbres XML sont de taille potentiellement grande (plus de 1000 nœuds). C'est pourquoi nous avons opté pour les CRFs, dont les algorithmes sont linéaires dans la taille des documents d'entrée.

Les travaux de [Gallinari et al., 2005] utilisent eux aussi un modèle stochastique pour apprendre à transformer des arbres XML. De la même façon que dans les CRFs, ce modèle permet de calculer la probabilité que le document XML de sortie ait été produit à partir

de l'arbre d'entrée. Toutefois, si cette méthode à permis d'obtenir de bons résultats, elle possède le désavantage de ne tirer que très peu parti de la structure de l'arbre d'entrée pour générer le document XML de sortie, à l'inverse de notre approche dans laquelle l'intégralité du document XML d'entrée (structure et contenu) vient conditionner le résultat.

On trouve aussi l'algorithme SEARN [Daumé III et al., 2006], qui ne se limite pas aux transformations d'arbres mais qui permet d'effectuer des tâches de prédiction de structures complexes, quelles qu'elles soient. Celui-ci est un méta-algorithme qui transforme le problème complexe de la prédiction de structures en plusieurs problèmes simples de classification multi-classes. Ainsi, il consiste en une succession de prédictions utilisant à chaque étape l'entrée et l'ensemble des prédictions précédentes. Bien que cet algorithme n'ait pas été utilisé précisément dans le cadre de la transformation d'arbres XML, il semble bien adapté. Toutefois, procédant itérativement, cette méthode est fortement dépendante de l'ordre de parcours des documents d'entrées, à l'inverse des CRFs dans lesquels une prédiction (une annotation) est considérée dans sa globabilité, et non un label après l'autre.

S'inspirant de l'algorithme SEARN et de LaSO [Daumé III and Marcu, 2005], l'algorithme ISM (*Incremental Structure Mapping*) [Maes et al., 2007] permet aussi d'effectuer des transformations d'arbres XML. Pour cela, il parcourt, dans l'ordre du document, les feuilles de l'arbre d'entrée et génère progressivement l'arbre de sortie. Pour chaque feuille, un certain nombre d'actions sont possibles (supprimer la feuille, l'insérer sous un nœud existant, créer de nouveaux nœuds pour l'insérer, *etc.*). L'arbre de sortie est complètement généré une fois l'ensemble des feuilles de l'arbre d'entrée parcourues. Cette méthode offre de bons résultats et possède, par rapport aux CRFs, l'avantage d'être rapide en phase de test, c'est-à-dire lors de la génération des arbres de sortie. Toutefois, la phase d'apprentissage est potentiellement très lente. De plus, comme pour SEARN, le résultat de la transformation est fortement dépendant de l'ordre de parcours des feuilles du document d'entrée.

Enfin, même s'il ne s'agit pas exactement de transformations d'arbres, les travaux de *Schema Matching* restent toutefois assez proches en s'intéressant au domaine de l'intégration de données XML. En effet, le but de cette tâche est d'apprendre automatiquement à partir d'exemples des correspondances entre les éléments des schémas XML d'entrée et de sortie afin de pouvoir interroger uniformément différentes sources de données (XML ou base de données relationnelle). Dans ce domaine, on distingue notamment les travaux de [Doan et al., 2001, Dhamankar et al., 2004] avec les systèmes LSD et iMAP. Le premier se concentre sur l'apprentissage de correspondance de type 1-1, c'est-à-dire des correspondances où à un élément du schéma source correspond exactement un élément du schéma cible. Nous montrons expérimentalement dans cette thèse que ce type de correspondances peut être découvert à l'aide des champs aléatoires conditionnels pour les arbres XML. Le système iMAP, pour sa part, permet aussi de découvrir des correspondances plus complexes où un élément cible correspond à la combinaison de plusieurs éléments sources (concaténation, multiplication de valeurs numériques, *etc.*). Notre approche ne permet pas de découvrir de telles correspondances.

4 Plan de la thèse

Dans le premier chapitre de cette thèse, nous étudierons comment il est possible de représenter une *transformation d'arbres* XML en les annotant. Pour cela, nous fournirons dans un premier temps un rappel des notions d'arbres XML. Puis, nous décrirons plus en détail les tâches de transformations d'arbres XML. Enfin, nous décrirons en quoi consiste l'annotation de tels arbres, avant de proposer deux techniques d'annotation pour la transformation.

Le chapitre 2 de cette thèse sera consacré aux *techniques d'apprentissage* à la fois pour l'annotation de séquences et d'arbres, et pour la transformations d'arbres. Les méthodes d'évaluation de ces techniques d'apprentissage y seront aussi abordées.

Le chapitre 3 s'intéressera pour sa part à la famille des *modèles graphiques* qui permettent de représenter des distributions de probabilité. Nous y présenterons dans un premier temps les différents types de modèles graphiques (dirigés et non dirigés) ainsi que l'algorithme d'inférence exacte qui leur est associé. Ensuite, nous aborderons le cas des modèles graphiques permettant d'effectuer des tâches d'annotation, en distinguant modèles génératifs et modèles conditionnels. Enfin, nous présenterons en détails le modèle des *champs aléatoires conditionnels* (ou CRFs) pour les séquences.

Dans le chapitre 4, nous présenterons notre *adaptation des champs aléatoires conditionnels* au cas de l'annotation d'arbres XML. Nous y décrirons trois modèles de dépendances différents pour les CRFs adaptés aux arbres, en les comparant. Puis, nous fournirons l'adaptation des *algorithmes d'apprentissage et d'annotation* pour ces modèles. Nous terminerons ce chapitre en comparant empiriquement les trois modèles de dépendances proposés sur des tâches d'annotation d'arbres XML.

Nous continuerons, avec le chapitre 5, par la présentation de deux techniques permettant d'améliorer la complexité des algorithmes pour les champs aléatoires conditionnels en utilisant des connaissances du domaine. Dans un premier temps, nous définirons comment ces connaissances offrent la possibilité d'introduire des *contraintes* dans les CRFs, réduisant ainsi le nombre d'annotations possibles pour une observation donnée. Puis, nous proposerons trois méthodes de *composition de champs aléatoires conditionnels*. Celles-ci permettent d'approximer un CRF sur l'ensemble des labels de la tâche d'annotation par plusieurs CRFs plus simples, définis chacun sur un sous-ensemble de labels. Ces deux techniques seront évaluées empiriquement en termes de résultats et de temps de calcul.

Enfin, dans le dernier chapitre, nous appliquerons les méthodes présentées dans les deux chapitres précédents à diverses *tâches de transformations d'arbres* XML à plus grande échelle. Ces tâches seront séparées en deux groupes selon que les arbres d'entrée sont dans un format orienté document, plus précisément XHTML, ou dans un format XML orienté données. Parmi ces tâches, nous mettrons en exergue l'une d'entre elles, consistant en l'apprentissage automatique de *générateurs de flux* RSS à partir de documents XHTML. Nous présenterons, dans ce sens, une application permettant à un utilisateur d'apprendre facilement de tels générateurs.

Chapitre 1

Transformations d'arbres XML par l'annotation

De plus en plus, le Web regorge d'informations, d'une utilité certes très variable, mais qu'il peut être intéressant de pouvoir réutiliser, par exemple dans le cadre de programmes informatiques. Ces informations ne sont toutefois pas facilement accessibles par de tels programmes. En effet, du fait de la grande diversité des sources de données du Web, celles-ci apparaissent dans des formats hétérogènes. Afin de pouvoir réutiliser ces informations, il est donc dans un premier temps nécessaire de les identifier et de les transformer dans un format connu.

Les données disponibles sur le Web étant au format XML, la majeure partie desquelles est au format XHTML, nous manipulons, lors de l'intégration de données du Web des arbres. De plus, le XML étant devenu un standard en termes d'échanges de données, il est préférable, pour garantir la réutilisabilité des données identifiées par des applications diverses, de les représenter sous la forme de documents XML dont la structure est connue. Il est donc nécessaire pour cela de savoir effectuer des tâches de transformation d'arbres XML. Nous proposons ici de modéliser ces tâches de transformation d'arbres par des tâches d'annotations.

Nous consacrons donc ce premier chapitre à ces deux tâches d'annotation et de transformation d'arbres XML. Pour cela, nous allons dans un premier temps rappeler quelques notions sur les arbres, tout particulièrement les arbres XML et leurs spécificités. Nous définirons ensuite les tâches de transformations d'arbres et plus précisément la transformation d'arbres XML pour l'intégration de données. Enfin, nous donnerons une courte introduction de l'annotation, avant de proposer deux façons de modéliser des tâches de transformations d'arbres XML par des tâches d'annotation d'arbres.

1.1 Notions d'arbres XML

Nous commençons donc ce chapitre par une définition des structures de données que nous allons manipuler : les arbres XML. Pour cela, nous rappelons d'abord quelques notions sur les arbres en général, avant de définir plus précisément ce que sont les arbres XML.

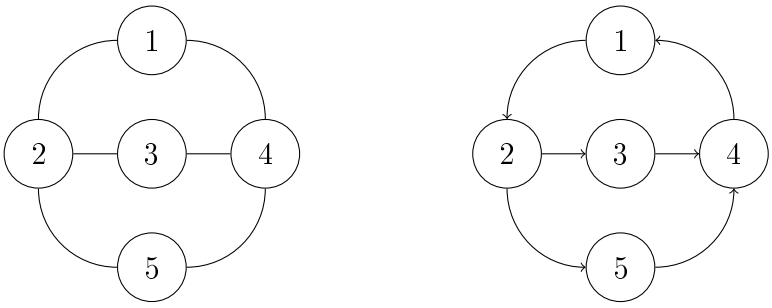

FIG. 1.1 – Exemples de graphes : (gauche) graphe non-orienté (droite) graphe orienté.

1.1.1 Définition des arbres

Il existe plusieurs façons de définir un arbre. Une première définition s'inscrit dans le domaine de la théorie des graphes. Il est toutefois nécessaire, avant de définir les arbres, de rappeler quelques définitions sur les graphes. Nous définissons tout d'abord les notions de graphe orienté et non-orienté.

Définition 1.1 *Un graphe orienté $\mathcal{G}$ est un couple (V, E) où :*
- *V est un ensemble dont les éléments constituent les **sommets** (aussi appelés **nœuds**) du graphe ;*
- *E est un ensemble de couples (ordonnés) de sommets appelés **arêtes**.*

Définition 1.2 *Un graphe non-orienté $\mathcal{G}$ est un couple (V, E) où :*
- *V est un ensemble dont les éléments constituent les **sommets** (aussi appelés **nœuds**) du graphe ;*
- *E est un ensemble de paires (non ordonnées) de sommets appelées **arêtes**.*

Un graphe non-orienté peut aussi être vu comme un graphe orienté tel que pour toute arête $(u, v) \in E$, il existe une arête $(v, u) \in E$.

La figure 1.1 montre deux exemples de graphes, l'un étant orienté et l'autre non. Pour les deux graphes, l'ensemble des sommets est $V = \{1, 2, 3, 4, 5\}$ et l'ensemble des arêtes est $E = \{(1, 2), (2, 5), (5, 4), (4, 1), (2, 3), (3, 4)\}$.

Nous définissons maintenant la notion de graphe connexe. Pour cela, nous rappelons tout d'abord la définition d'un chemin dans un graphe. Un chemin d'un graphe $\mathcal{G} = (V, E)$ est une séquence de sommets $(v_0, \ldots, v_l)$ telle que pour tout $i < l$, $(v_i, v_{i+1}) \in E$. À l'aide de cette définition, nous pouvons maintenant définir ce qu'est un graphe connexe. Un graphe $\mathcal{G} = (V, E)$ est **connexe** si et seulement si pour tous sommets u et v de V, il existe un chemin de u à v. Les deux graphes de la figure 1.1 sont connexes.

Nous définissons maintenant la notion de cycle. Dans un graphe $\mathcal{G} = (V, E)$, un **cycle** de longueur l est une séquence de sommets $(v_0, \ldots, v_l)$ distincts à l'exception de v_0 et v_l telle que pour tout $i < l$, $(v_i, v_{i+1}) \in E$. On appelle **graphe acyclique** un graphe ne possédant pas de cycle. Les deux graphes de la figure 1.1 ne sont pas acycliques. En effet, ils possèdent de nombreux cycles, parmi lesquels $(1, 2, 5, 4, 1)$.

Ces notions nous permettent de définir un arbre.

Définition 1.3 *Un **arbre** est un graphe non orienté acyclique et connexe.*

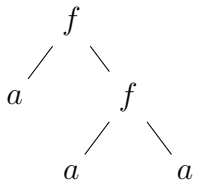

FIG. 1.2 – Premier exemple d'arbre.

Comme c'est le cas des arbres que nous allons considérer par la suite, nous définissons aussi la notion d'arbre enraciné. Un **arbre enraciné** est arbre (V, E) dans lequel un nœud particulier $r \in V$ est désigné comme étant la **racine**. Cette racine r est le seul nœud de l'arbre tel qu'il n'existe pas d'arête de la forme (u, r) où $u \in V$.

Une seconde définition logique des arbres peut venir se greffer sur celle que nous venons de donner. Soit un alphabet Σ constitué de symboles de fonctions et de constantes. Un arbre t sur l'alphabet Σ est défini par :

- $\texttt{nodes}(t) \subset \mathbb{N}^*$ est l'ensemble des nœuds de l'arbre. Chaque nœud est représenté par un mot sur l'ensemble $\mathbb{N}^*$ des n-uplets d'entiers naturels décrivant le chemin depuis la racine. La racine est représentée par le mot vide ϵ. Le nœud 2.1 correspond au premier fils du second fils de la racine. L'ensemble $\texttt{nodes}(t)$ est clos par préfixe, c'est-à-dire que pour tout mot $a \in \texttt{nodes}(t)$, tout préfixe b de a appartient à $\texttt{nodes}(t)$. Le nœud représenté par le mot b est alors un ancêtre du nœud représenté par a. L'ensemble $\texttt{nodes}(t)$ correspond à l'ensemble V des sommets du graphe dans la définition précédente.
- une relation $\texttt{child} : \texttt{nodes}(t) \rightarrow \texttt{nodes}(t)$ tel que si $\texttt{child}(a) = b$, b est appelé *fils* de a, a étant le père de b. La racine ϵ est le seul nœud qui ne possède pas de père, c'est-à-dire $\nexists a, \texttt{child}(a) = \epsilon$. Cette relation implique la propriété suivante :

  $$\forall a, b \in \texttt{nodes}(t),\ \texttt{child}(a) = b \Rightarrow b = a.i \text{ avec } i \in \mathbb{N}$$

  L'ensemble des couples (a, b) tels que $\texttt{child}(a) = b$ correspond à l'ensemble E des arêtes du graphe.
- une fonction d'étiquetage $\texttt{etiq} : \texttt{nodes}(t) \rightarrow \Sigma$ qui associe à chaque nœud de l'arbre une étiquette dans l'alphabet Σ des symboles.

Sur l'exemple de la figure 1.2, l'alphabet des symboles est $\Sigma = \{f, a\}$ et l'ensemble des nœuds est $\texttt{nodes}(t) = \{\epsilon, 1, 2, 2.1, 2.2\}$. L'étiquette de la racine est $\texttt{etiq}(\epsilon) = f$, et le nœud 2.1 est étiqueté par $\texttt{etiq}(2.1) = a$.

L'**arité** est une fonction de Σ dans $\mathbb{N}$. Un symbole f de l'alphabet Σ est d'arité n si le nombre de fils d'un nœud étiqueté par f est n. Sur notre exemple, l'arité du symbole f est 2. Les symboles d'arité 0 forment les étiquettes des feuilles de l'arbre et sont appelés **constantes**, tandis que les symboles d'arité supérieure à 0 sont appelés symboles de **fonctions**. On dit qu'un arbre est d'arité fixe si tous ses symboles de fonctions possèdent la même arité. L'exemple de la figure 1.2 est un arbre d'arité 2 (arbre binaire). À l'inverse, un arbre dont les symboles de fonction peuvent avoir un nombre quelconque de fils est appelé arbre d'arité non bornée.

La définition d'un arbre que nous venons de donner concerne les **arbres non ordonnés**, c'est-à-dire les arbres pour lesquels il n'existe pas de relation d'ordre sur l'ensemble

```
<movie year="1998">
  <title>Pi</title>
  <director>
    <firstname>Darren</firstname>
    <name>Aronofsky</name>
  </director>
</movie>
```

FIG. 1.3 – Exemple de document XML décrivant un film.

des fils d'un même nœud. Toutefois, il est possible de définir un ordre partiel ou total sur cet ensemble. Cet ordre est défini par la relation `next-sibling` : `nodes`$(t) \rightarrow$ `nodes`(t). `next-sibling`$(a) = b$ signifie que b est le frère suivant de a. Sur la figure 1.2, on a la relation `next-sibling`$(2.1) = 2.2$. Si cette relation est définie pour tous les fils d'un même nœud, on dit que l'arbre est **ordonné**, tandis que quand la relation `next-sibling` n'est pas définie pour tous les fils d'un même nœud, on dit que l'arbre est **partiellement ordonné**.

1.1.2 Les arbres XML

Le format XML, pour *eXtensible Markup Language* (langage de balisage extensible) est devenu ces dernières années le standard incontournable en termes d'échanges de données, particulièrement sur le Web. De nombreux langages et outils recommandés par le World Wide Web Consortium[2] (W3C) permettent de typer, transformer et effectuer des requêtes sur les documents XML. Nous définissons maintenant la notion d'arbre XML ainsi que les différents types de documents XML que nous allons rencontrer dans nos applications.

Définitions des arbres XML

Les documents XML sont des représentations textuelles de données ayant une structure hiérarchique. La figure 1.3 est un exemple de document XML décrivant un film. Les informations contenues dans ce document sont organisées hiérarchiquement. La racine du document correspond à la balise `film`. Ce film possède un titre, une année et un réalisateur, le nom de ce dernier étant décomposé en prénom et nom. Les documents XML peuvent être représentés sous forme d'arbres dont les feuilles contiennent les données textuelles, comme le montre la figure 1.4 qui est la représentation arborescente de l'exemple de la figure 1.3. Le W3C a mis au point une représentation arborescente standardisée pour les documents XML appelée DOM (Document Object Model). Dans la suite du document, nous ferons toujours référence à cette représentation lorsque nous parlerons d'arbre XML. Dans cette représentation, les arbres ont plusieurs types de nœuds. Parmi eux, on compte trois types principaux : les éléments, les attributs et les nœuds texte. Les éléments correspondent aux nœuds internes de l'arbre XML (les **balises**). Sur l'exemple de document XML de la figure 1.3, les nœuds `movie`, `title`, `director` sont des éléments. Les attributs sont des

[2] `http://www.w3.org`

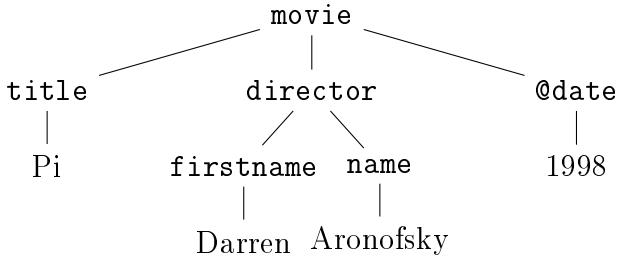

FIG. 1.4 – Arbre XML décrivant un film.

nœuds particuliers venant fournir des informations supplémentaires aux éléments. Dans notre exemple, le nœud `year` est un attribut, fils de l'élément `movie`. Enfin, les nœuds texte contiennent les données textuelles du document, telles que "Pi", ou "Darren" dans notre exemple. La figure 1.4 est la représentation arborescente du document XML de la figure 1.3 dans le standard DOM. Par convention, nous distinguons les attributs des autres fils d'un nœud en ajoutant le préfixe "@" à leur étiquette. Les éléments d'un document XML pouvant avoir un nombre quelconque de fils, les arbres XML sont donc des arbres d'arité non bornée. De plus, ces arbres sont partiellement ordonnés. En effet, les éléments et les nœuds texte sont ordonnés selon la relation `next-sibling` telle que définie dans la section 1.1.1. Par contre, il n'existe pas d'ordre sur les attributs. Ainsi, un élément a un ensemble de fils ordonnés de type élément ou texte, et un ensemble de fils non ordonnés de type attributs.

Typage des arbres XML

Les arbres XML pouvant prendre des formes très variées, il est possible, voire souvent nécessaire, de les typer au moyen de langages de description de documents XML ou *Document Schema Definition Languages* (DSDL). Les DSDL sont des langages déclaratifs qui, par un ensemble de règles, permettent de définir le schéma d'un document XML. On trouve, parmi ces langages de schémas les DTD, XML Schema ou encore RelaxNG [Bex et al., 2004]. Les schémas ont pour but de contraindre la structure des arbres XML, ainsi que les étiquettes associées aux nœuds éléments et attributs, dans l'optique de la compatibilité avec une application ou d'un échange standardisé d'informations. Un exemple de typage d'arbre XML est la DTD de XHTML, dédiée aux applications hypertextes. Celle-ci garantit d'avoir des arbres XML que les navigateurs Web sont capables d'interpréter, leur permettant d'afficher au mieux les pages Web.

Le langage de schémas le plus courant et aussi le plus basique est la DTD, pour *Document Type Definition* (Définition de Type de Document). Une DTD indique les noms des éléments pouvant apparaître dans les documents XML qu'elle décrit, ainsi que leur contenu, c'est-à-dire les sous-éléments, les attributs et le contenu textuel. La figure 1.5 présente un exemple de DTD. Sur cet exemple, on remarque que les éléments acceptés sont `movie`, `title`, `director`, `firstname` et `name`. La définition du contenu d'un élément est représenté par une expression régulière. Par exemple, les règles du type `ELEMENT` indiquent qu'un élément `movie` a comme fils un élément `title` et un ou plusieurs éléments `director`, tandis qu'un élément `director` contient un élément `name`, éventuellement précédé d'un

```
<!ELEMENT movie title,director+>
<!ATTLIST movie year CDATA>
<!ELEMENT title #PCDATA>
<!ELEMENT director firstname?,name>
<!ELEMENT firstname #PCDATA>
<!ELEMENT name #PCDATA>
```

FIG. 1.5 – Exemple de DTD pour les documents XML décrivant un film (*cf.* figure 1.3).

élément `firstname`. Enfin, la règle `ATTLIST` précise qu'un élément `movie` possède un attribut `year`. L'exemple de document XML de la figure 1.3 est valide selon cette DTD. Les DTDs restent toutefois très limitées. En effet, elles ne permettent pas de poser des contraintes sur le contenu textuel d'un élément. Dans notre exemple, il n'est pas possible de préciser que le contenu textuel de l'attribut `year` est un entier de 4 chiffres. De plus, dans les DTDs, les types d'éléments sont identifiés par leur nom, quel que soit leur contexte d'apparition dans l'arbre. Par exemple, il n'est pas possible de définir qu'un élément nommé `director` puisse ne contenir que du texte et non une décomposition en `firstname` et `name` dans un contexte différent (lorsqu'il n'est pas fils d'un élément `movie`).

Afin de permettre des définitions de schémas plus précis qu'avec les DTDs, des langages de schémas plus avancés comme XML Schema [Vlist, 2002] et Relax NG [Vlist, 2003] ont été créés. Ceux-ci permettent ainsi dans un premier temps de contraindre le contenu autorisé dans les nœuds texte de l'arbre, à l'aide de types prédéfinis, ou à l'aide de la définition de nouveaux types par l'utilisateur, comme par exemple un type `année` correspondant à un entier de 4 chiffres. Ces langages de schémas permettent aussi de fournir des définitions différentes à des éléments de même nom selon le contexte dans lequel ceux-ci apparaissent.

Familles d'arbres XML

D'un point de vue général, on distingue deux grandes familles de documents XML selon leur typage. Dans certains cas, le typage des documents XML peut servir à exprimer des informations sémantiques concernant le contenu textuel du document. C'est le cas de notre premier exemple sur la figure 1.4. En effet, cet arbre contient des informations de type "base de données" sur un film. La structure de l'arbre et les étiquettes associées aux nœuds nous informent que le titre du film est "Pi", que l'année du film est "1998" et que son réalisateur est "Darren Aronofsky". Dans la communauté XML, les documents de ce type sont couramment appelés documents XML **orientés données**. Ce type de documents contient des étiquettes très informatives et le contenu des nœuds texte est souvent relativement court.

À l'inverse, dans le cas du XHTML, les étiquettes des nœuds expriment plutôt une information sémantique sur la structure du document ainsi que sur sa présentation, c'est-à-dire la façon de l'afficher dans un navigateur. Ces étiquettes ne fournissent par contre aucune information sur la sémantique des données contenues dans les documents XHTML. On dit alors que ces documents XML sont **orientés document**. C'est le cas de l'arbre XHTML représenté sur la figure 1.6. Les étiquettes des nœuds de cet arbre (`h1`, `h2`, `p`)

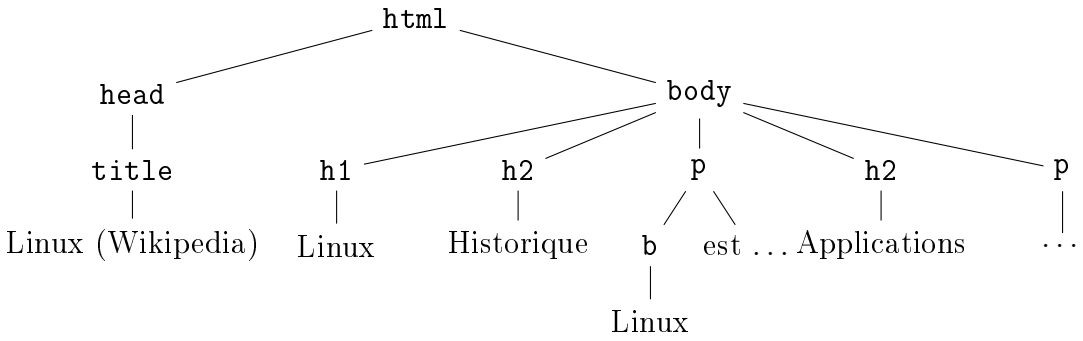

FIG. 1.6 – Exemple d'arbre XHTML : XML orienté document.

expriment bien une information sur la structure du document, `h1` correspondant à un titre principal, `h2` à un titre de sous-partie et `p` à un paragraphe. Par contre, aucune information sur la sémantique du contenu textuel du document n'est fournie. Plusieurs formats XML classiques rentrent dans cette catégorie de XML orienté document : XHTML, TEI (Text Encoding Initiative) ou encore DOCBOOK.

1.2 Définition des tâches

Maintenant que nous avons défini les données avec lesquelles nous allons travailler, nous allons décrire les tâches que nous abordons. Pour cela, nous allons dans un premier temps présenter un exemple de tâche d'intégration de données, pour ensuite nous intéresser plus précisément à la transformation d'arbres XML.

1.2.1 Intégration de données

Depuis l'arrivée du Web, de plus en plus de sources d'informations sont à notre disposition. Toutefois, de par la variété des sources, ces informations se retrouvent dans des formats hétérogènes qui les rendent très difficiles, voire même impossibles, à réutiliser dans le cadre, par exemple, de programmes informatiques. L'**intégration de données** consiste à permettre cette réutilisation en identifiant parmi ces sources les informations pertinentes et en les intégrant dans un format compréhensible par les programmes informatiques (bases de données, XML, *etc.*). Ce format est souvent appelé **schéma médiateur**. De plus, pour permettre la communication entre plusieurs applications, le format XML est souvent utilisé. Toutefois, afin de pouvoir utiliser les informations qui lui sont envoyées, une application doit connaître la structure du document XML. Il est alors nécessaire de transformer les données XML envoyés afin de leur faire suivre un schéma connu de l'application. L'intérêt de la transformation d'arbres XML est donc multiple.

Le domaine de l'**extraction d'informations** a fourni de nombreuses méthodes permettant d'effectuer des tâches d'intégration de données et des transformations très simples dans lesquelles les documents ne sont que peu structurés. En effet, l'extraction d'informations a pour but d'extraire des informations structurées, c'est-à-dire des informations dont la sémantique est connue dans un domaine précis, à partir de documents ne permettant

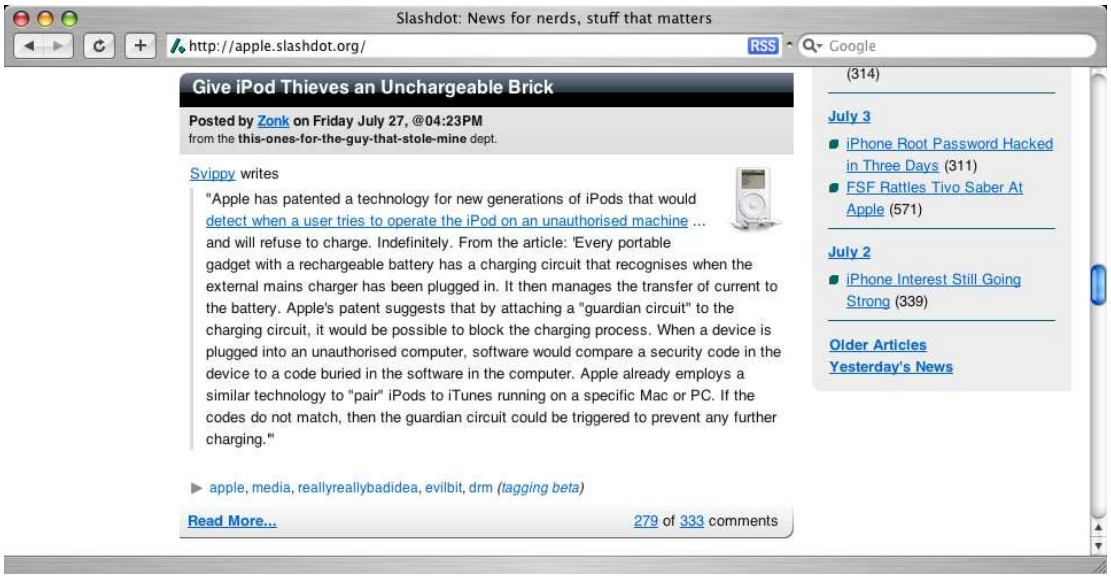

FIG. 1.7 – Page Web issue du site Slashdot.

pas, a priori, de connaître la sémantique des informations qu'ils contiennent. Une tâche courante de ce domaine est la reconnaissance d'entités nommées, qui consiste à identifier dans un document, les noms d'entités (personne, organisme, lieu, *etc.*), les dates ou encore les expressions de durée. De nombreux systèmes, parmi lesquels Minorthird[3], permettent de résoudre cette tâche.

Nous illustrons maintenant par un exemple le type de tâches de transformation d'arbres XML que nous abordons ici. Nous considérons donc la tâche suivante : nous voulons pouvoir réutiliser les informations contenues dans la page Web de la figure 1.7. Pour cela, nous choisissons de les représenter sous la forme de flux RSS. Le format RSS, pour *Rich Site Summary* ou *Really Simple Syndication*, est un format XML pour la syndication de contenu Web, utilisé pour diffuser des mises à jour de sites dont le contenu est susceptible de changer régulièrement, comme par exemple les sites d'information. Ce format est lisible par de nombreuses applications dédiées au RSS ou par la majeure partie des navigateurs Web. Ce format XML liste différentes informations, représentées par un élément `item`, contenant chacune un titre, une description et éventuellement l'adresse de la page correspondant à la news, son auteur et sa date de publication. La figure 1.7 présente une information, dont le titre est "Give iPod Thieves an Unchargeable Brick", l'auteur est "Zonk", la date de publication "Friday July 27" et la description est le grand paragraphe commençant par "Apple has...".

Pour résoudre cette tâche, plusieurs approches sont possibles. Dans un premier temps, le choix de la représentation de la page Web d'entrée est primordial. En effet, de nombreuses méthodes d'extraction d'informations choisissent de considérer cette page comme une séquence de mots, parmi lesquels on souhaite identifier le titre, l'auteur, *etc.* Pour les identifier, il suffit de remarquer que, par exemple, le nom de l'auteur se trouve entre le texte "Posted by" et le mot "on", et qu'il est suivi de la date de publication.

[3] `http://minorthird.sourceforge.net/`

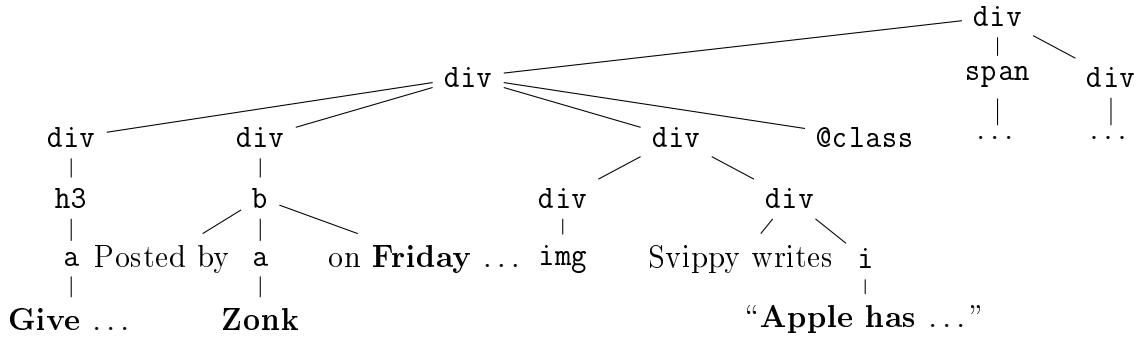

FIG. 1.8 – Arbre XHTML correspondant à la page Web de la figure 1.7. Les parties à identifier sont représentées en gras.

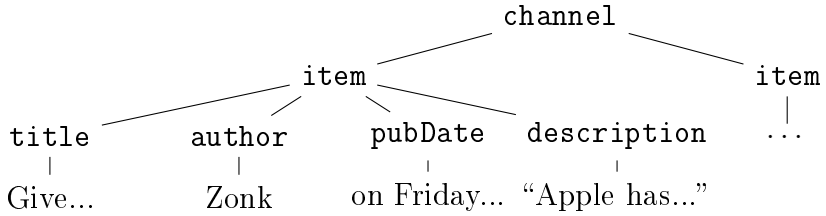

FIG. 1.9 – Portion de l'arbre RSS correspondant à la page Web de la figure 1.7.

Si elles peuvent être efficaces, nous considérons tout de même que ces méthodes ne tirent pas complètement parti de la donnée qui leur est fournie en entrée. En effet, une page Web est un document XHTML. Celui-ci est donc un arbre XML dont la structure donne des informations sur sa mise en page. La plupart des pages Web existantes étant générées automatiquement par des programmes informatiques, leur structure possède des régularités permettant de simplifier l'identification des données qu'elles contiennent. Nous choisissons donc, à l'inverse des méthodes citées précédemment, de représenter les pages Web par leur arbre DOM. La figure 1.8 montre cette représentation de la page Web de la figure 1.7. On remarque, sur cet arbre, que les différentes informations sont aisément identifiables : le titre est la première feuille texte, l'auteur est le fils du deuxième fils du nœud `b` et la description est dans un nœud `i`.

En considérant la page Web d'entrée comme un arbre, et sachant que le flux RSS que l'on veut obtenir en sortie est lui aussi un arbre XML, représenté en figure 1.9, la tâche présentée ici correspond donc bien à une tâche de transformation d'arbres XML.

1.2.2 Transformation d'arbres XML

Nous décrivons maintenant plus en détails le principe de la transformation d'arbres XML. Celle-ci a pour but de transformer des arbres XML dont le schéma ou la structure sont a priori inconnus en arbres XML suivant un schéma établi et interprétables par une application. Plus formellement, on peut reformuler une telle tâche de transformation comme suit : soient S et T deux schémas XML, respectivement un schéma source et un schéma cible. Soit $\mathbf{x}$ un arbre XML suivant le schéma S. Une tâche de transformation consiste

à transformer cet arbre **x** en un arbre XML **y** représentant les mêmes données suivant le schéma cible T. Il est à noter que, selon les tâches de transformation considérées, le schéma cible T n'est pas nécessairement connu a priori.

Cette tâche a déjà été abordée dans les travaux sur le *Schema Matching*, notamment ceux de [Doan, 2002, Doan et al., 2003]. Le *Schema Matching* constitue néanmoins une tâche assez différente de celle de la transformation. En effet, cette tâche s'intéresse aux schémas des arbres XML et non aux documents en eux-mêmes. Ainsi, une tâche de *Schema Matching* consiste, étant donnés les deux schémas S et T, respectivement la source et la cible, ainsi qu'un ensemble de documents suivant le schéma source S, à trouver des correspondances entre les étiquettes des nœuds du schéma source et celles du schéma cible. Ces correspondances (ou *mappings*) sont souvent très simples. Le cas le plus simple est celui des correspondances 1-1. Dans ce cas, la tâche de *Schema Matching* consiste à trouver, pour chaque élément s du schéma S, l'élément t du schéma T qui est, sémantiquement, le plus proche possible de s. La figure 1.10 présente un exemple de *Schema Matching*, extrait du corpus "Real Estate I" constitué par Anhai Doan [Doan, 2002]. On remarque que sur cet exemple, la plupart des correspondances sont des correspondances 1-1. C'est le cas par exemple de l'élément `firm` dans le schéma source qui devient `firm_info` dans le schéma cible, ou encore de `location` qui devient `firm_address`. Toutefois, des correspondances plus complexes peuvent exister. Par exemple, l'élément `adresse` dans un schéma cible peut correspondre à la concaténation des éléments `rue` et `ville` d'un schéma source.

Toutefois, la tâche de transformation complète ne se résume pas à cela et est légèrement plus complexe que le *Schema Matching*. En effet, une fois les correspondances entre labels trouvées, l'arbre de sortie suivant le schéma cible T doit alors être reconstitué. Pour effectuer de telles transformations, les correspondances 1-1 constituant la majeure partie du *Schema Matching* ne suffisent pas. En effet, dans l'arbre d'entrée, deux nœuds ayant la même étiquette peuvent avoir des étiquettes différentes dans l'arbre de sortie. C'est le cas sur l'exemple de la figure 1.10, où l'étiquette `name` de l'arbre d'entrée a deux étiquettes correspondantes différentes dans l'arbre de sortie : `firm_name` et `agent_name`. La seule étiquette ne permet donc pas de différencier les deux correspondances présentes dans cet exemple, il est aussi nécessaire de considérer le contexte dans lequel cette étiquette apparaît. Il paraît donc, pour tenir compte du contexte dans les transformations, plus naturel de travailler directement sur les données, et non sur les schémas, afin d'obtenir des méthodes de transformation plus efficaces.

Les méthodes de transformation que nous proposons sont donc directement basées sur les arbres XML que nous voulons transformer. Ces méthodes consistent à annoter les arbres XML, c'est-à-dire affecter un label à chaque nœud des arbres, cette annotation définissant une transformation de l'arbre d'entrée dans le schéma source en un arbre de sortie dans le schéma cible. Avant de présenter ces méthodes de transformation par l'annotation, nous rappelons dans un premier temps en quoi consiste une tâche d'annotation.

1.3 Qu'est-ce que l'annotation ?

Nous définissons donc maintenant ce qu'est une tâche d'annotation. Pour cela, nous partons tout d'abord des bases de l'annotation, c'est-à-dire la classification, afin de repla-

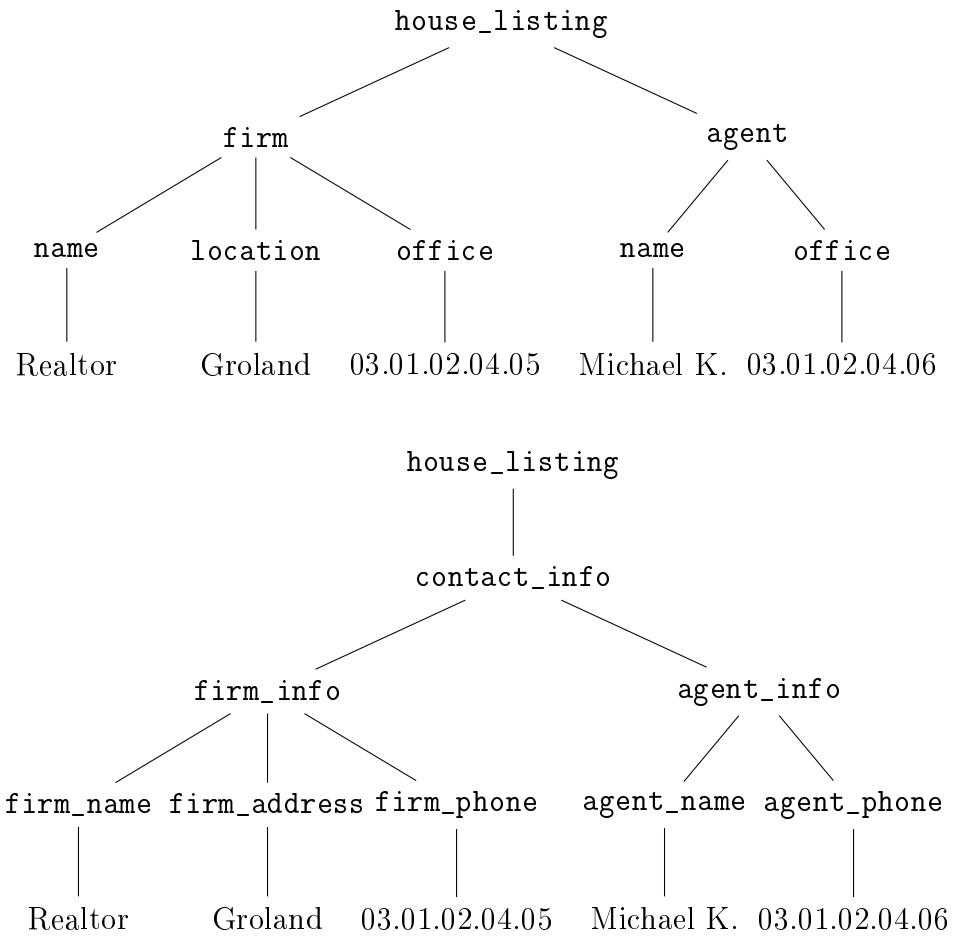

FIG. 1.10 – Exemple de Schema Matching : (haut) Arbre XML d'entrée suivant le schéma source (bas) Arbre XML suivant le schéma cible. Les nœuds `name` peuvent devenir `agent_name` ou `firm_name` selon le contexte.

cer l'annotation dans son contexte. Nous illustrons ensuite avec différentes applications possibles de l'annotation dans le cas des séquences et des arbres.

1.3.1 De la classification à l'annotation

Une tâche d'annotation peut toujours être considérée comme une tâche de transformation simple où l'entrée, que nous noterons $\mathbf{x}$, est l'observation que l'on veut annoter, et son annotation est une sortie $\mathbf{y}$ de même structure, et donc de même taille. Dans le cas des transformations, $\mathbf{x}$ et $\mathbf{y}$ sont des vecteurs, respectivement de taille n_x et n_y. Pour tout $i \in [1, n_x]$, la i^{eme} composante de $\mathbf{x}$ x_i appartient à l'alphabet d'entrée $\mathcal{X}$. De la même façon, pour tout $i \in [1, n_y]$, y_i représente la i^{eme} composante de $\mathbf{y}$, et on a $y_i \in \mathcal{Y}$, où $\mathcal{Y}$ est appelé l'**alphabet des labels**.

Dans ce contexte, on peut déjà noter que le cas le plus simple d'une tâche d'annotation est celui de la **classification binaire**. En effet, une tâche de classification binaire consiste à associer à une observation $\mathbf{x}$ de taille quelconque une classe y, telle que $y \in \{0, 1\}$. Dans une telle tâche de classification binaire, la sortie $\mathbf{y}$ n'est donc pas structurée, et sa taille $n_y = 1$. De plus, l'alphabet des labels $\mathcal{Y}$ est restreint à l'ensemble $\{0, 1\}$. On trouve des applications de la classification binaire dans de nombreux domaines tels que :

- le domaine médical : la classification de données médicales permet de déterminer si un patient est atteint ou non d'une maladie.
- la recherche d'informations : la classification binaire de pages Web, selon que celles-ci contiennent ou non un ensemble de mots, permet par exemple de déterminer si une page doit faire partie du résultat d'une recherche.
- la classification de courriers électroniques en courrier désirable ou indésirable.

Les tâches de classification binaire ont, par la suite, été étendues afin de ne plus se limiter à deux classes, donnant ainsi lieu à la **classification multi-classes**. Une telle tâche consiste à attribuer à une observation x une classe y telle que $y \in \mathcal{Y}$, où $\mathcal{Y}$ correspond à l'ensemble des classes définies dans la tâche. Une des applications les plus connues de la classification multi-classes est la tâche de reconnaissance de caractères ou *Optical Character Recognition* (OCR), mais cette tâche trouve aussi des applications dans de nombreux domaines tels que le domaine médical, la bioinformatique ou encore l'extraction d'informations.

Enfin, l'évolution la plus récente est celle de la classification structurée multi-classes. Communément appelée *Structured Output* (pour **sortie structurée**), cette tâche consiste à associer à une entrée $\mathbf{x}$ structurée, une sortie $\mathbf{y}$, elle aussi structurée, mais ne possédant pas forcément la même structure que $\mathbf{x}$. Les tâches de transformation d'arbres font partie de ce domaine.

Dans ce domaine de la classification structurée multi-classes, les tâches d'annotation constituent un cas particulier. En effet, dans le cas de l'annotation, la structure de la sortie $\mathbf{y}$ est la même que celle de l'observation $\mathbf{x}$ fournie en entrée. En d'autres termes, à chaque composante x_i de l'observation est associée une classe ou un label y_i dans l'annotation, ces classes étant structurées entre elles de la même façon que dans l'observation.

Définition 1.4 *Un tâche d'annotation consiste à associer à une observation* $\mathbf{x}$ *une annotation* $\mathbf{y}$*, telle que* $\mathbf{y}$ *possède la même structure sous-jacente que* $\mathbf{x}$ *et telle que à chaque*

composante x_i de $\mathbf{x}$ *est associée la classe (ou le label)* y_i *de* $\mathbf{y}$.

Nous décrivons dans les sections suivantes deux types de tâches d'annotation, ainsi que leurs applications possibles, pour deux structures de données différentes : les séquences et les arbres.

1.3.2 Annotation de séquences

La tâche d'annotation dans le cas des séquences peut être définie comme suit. Soit une séquence $\mathbf{x} = x_1 \dots x_n$ de taille n telle que $\forall 1 \leq i \leq n,\ x_i \in \mathcal{X}$. Annoter la séquence $\mathbf{x}$ selon l'alphabet de labels $\mathcal{Y}$ consiste à produire en sortie une séquence $\mathbf{y} = y_1 \dots y_n$ de même taille que la séquence $\mathbf{x}$, telle que $\forall 1 \leq i \leq n,\ y_i \in \mathcal{Y}$.

Pour illustrer une tâche d'annotation de séquences, on considère l'exemple de page Web de la figure 1.7, à partir de laquelle on veut générer un flux RSS. On représente ici cette page comme une séquence de mots de longueur n. La tâche d'annotation consiste donc à associer à chaque mot x_i de cette page un label y_i, selon que celui-ci fait partie du titre, du nom de l'auteur, de la date de publication, de la description ou d'aucune de ces 4 catégories. Ainsi, dans ce cas, l'alphabet des labels $\mathcal{Y}$ est $\{\texttt{titre}, \texttt{auteur}, \texttt{pubDate}, \texttt{description}, 0\}$. Chaque mot du titre "Give iPod Thieves an Unchargeable Brick" doit donc être annoté par `titre`, "Zonk" par `auteur`, *etc.*

L'annotation de séquences trouve de nombreuses applications dans différents domaines, parmi lesquels le Traitement Automatique des Langues Naturelles (TALN) et l'Extraction d'Informations.

Le Traitement Automatique des Langues Naturelles est un domaine de recherche mêlant informatique et linguistique. Ce domaine s'attache à développer des programmes informatiques permettant d'aider à comprendre le langage humain, de façon à aider sa compréhension par un ordinateur, dans le but de réutiliser ces informations, essentiellement dans d'autres programmes. Les phrases pouvant être considérées comme des séquences de mots (et les textes, des séquences de phrases), de nombreux problèmes dans ce domaine correspondent donc à des tâches d'annotations de séquences. Une première tâche d'annotation du TALN est la tâche d'annotation des catégories morpho-syntaxiques (parties de discours), plus communément appelée *Part-Of-Speech tagging* [Jelinek, 1985, Brill, 1993]. Cette tâche consiste à associer à chaque mot d'une phrase sa catégorie morpho-syntaxique, par exemple "déterminant", "verbe", ou encore "nom commun". La figure 1.11 montre un exemple d'une telle tâche. Sur cette exemple, le label `DT` indique que "Le" est un déterminant, `NN` indique que "chien" est un nom commun, tandis que "court" est annoté comme étant un verbe (`VB`). On trouve aussi dans le domaine du TALN la tâche de segmentation de phrases ou *Noun Phrase Chunking*. Celle-ci consiste à séparer des phrases en plusieurs portions (groupes nominaux, groupes verbaux) qui ne se recoupent pas. Cette tâche peut être considérée comme une tâche d'annotation des séparateurs entre les mots de phrases, revenant donc à trouver pour ces phrases un parenthésage correct.

L'annotation de séquences trouve aussi de nombreuses applications en extraction d'informations. Le but de l'extraction d'informations est d'extraire automatiquement des informations structurées à partir de documents pas ou peu structurés, tels que des textes en langage naturel. Une première application se trouve dans le domaine de l'extraction

x :	Le	chien	court.
y :	DT	NN	VB

FIG. 1.11 – Exemple d'annotation *part-of-speech*. La première ligne est l'observation, la seconde est son annotation.

x :	Mozart	est	né	en	1756.
y :	0	0	0	0	1

FIG. 1.12 – Exemple d'annotation pour l'extraction d'une date de naissance.

monadique dans les séquences, c'est-à-dire l'extraction d'un seul type d'information, par exemple des adresses électroniques, ou encore des noms de personne. Pour cela, l'alphabet des labels $\mathcal{Y}$ est constitué de deux labels : 1 pour les données à extraire et 0 pour les autres. La figure 1.12 montre un exemple d'extraction de dates de naissance. Sur cet exemple, la date de naissance à extraire (celle de Mozart), est "1756" et elle est annotée par 1. Une autre application majeure dans le domaine de l'extraction d'informations est la reconnaissance d'entités nommées [Sang and Meulder, 2003] qui consiste à identifier, dans du texte, des noms de personnes, de lieux ou encore d'organismes. Cette application est typiquement une tâche d'annotation, chaque entité nommée étant annotée par son type.

1.3.3 Annotation d'arbres

Nous définissons maintenant ce qu'est une tâche d'annotation dans le cas des arbres. Annoter un arbre d'entrée $\mathbf{x}$ dont les étiquettes des nœuds prennent leurs valeurs dans $\mathcal{X}$ consiste à produire un arbre $\mathbf{y}$, de même structure que $\mathbf{x}$, mais dont les étiquettes sont définies dans l'alphabet des labels $\mathcal{Y}$. Cela revient, en d'autres termes, à associer à chaque nœud de l'arbre d'entrée un label appartenant à l'alphabet $\mathcal{Y}$.

Nous illustrons une annotation d'arbre en reprenant l'exemple de page Web de la figure 1.7 et sa représentation arborescente 1.8. Dans le cadre de la tâche d'intégration de données définie dans la section 1.2.1, on souhaite identifier le titre, l'auteur, la date et la description. Pour cela, on associe à chaque nœud de l'arbre de la figure 1.8 un label parmi $\{\texttt{titre}, \texttt{auteur}, \texttt{pubDate}, \texttt{description}, \perp\}$, $\perp$ étant associé aux nœuds ne contenant aucune de ces informations. L'annotation qui en résulte est représentée sur la figure 1.13.

Le domaine de l'annotation d'arbres a été nettement moins étudié que celui de l'annotation de séquences. On trouve tout de même, particulièrement en Traitement Automatique des Langues Naturelles, quelques tâches d'annotation d'arbres, parmi lesquelles la tâche d'annotation des rôles sémantiques, ou *Semantic Role Labeling*, de CONLL 2004 et 2005 [Carreras and Marquez, 2005]. En TALN, un rôle sémantique est la relation qui lie un constituent, c'est-à-dire un mot ou un groupe de mot, à un prédicat, typiquement le verbe. Par exemple, dans la phrase "Il offre un cadeau à sa sœur", le prédicat est "offre" et les trois rôles sémantiques sont :

– "Il" : le donneur.
– "un cadeau" : l'objet donné.

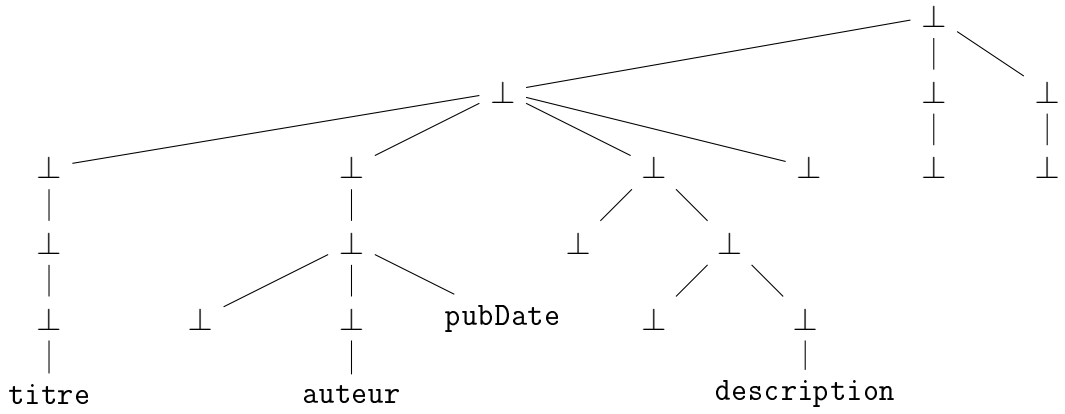

FIG. 1.13 – Annotation de l'arbre de la figure 1.8 avec l'alphabet $\{\texttt{titre}, \texttt{auteur}, \texttt{pubDate}, \texttt{description}, \bot\}$, $\bot$ correspond à l'absence d'information.

– "sa sœur" : le receveur.

Si cette tâche peut correspondre à une tâche d'annotation de séquences, elle est toutefois souvent modélisée comme une tâche d'annotation d'arbres, les arbres annotés étant les arbres d'analyse syntaxique des phrases.

Dans le cadre du projet de Web Sémantique[4], une autre tâche d'annotation d'arbres est l'annotation sémantique de pages Web. Cette tâche consiste à affecter aux nœuds de l'arbre des labels sémantiques provenant d'une ontologie, c'est-à-dire une hiérarchie de concepts. L'identification des informations et de leurs titres et auteurs dans la page Web de la figure 1.7 s'inscrit par exemple dans le cadre de l'annotation sémantique.

1.4 Annoter des arbres XML pour les transformer

Maintenant que nous avons défini ce qu'est l'annotation d'arbre, et dans quels domaines celle-ci est le plus souvent mise en œuvre, nous nous intéressons à l'application que nous en faisons. Nous présentons donc ici comment nous utilisons l'annotation d'arbres pour modéliser des transformations d'arbres XML. Le principe est le suivant : il est nécessaire de définir une façon d'annoter les arbres d'entrée de manière à ce que leur annotation définisse une unique transformation. Pour cela, plusieurs approches sont possibles. Nous en proposons deux, l'une consiste à annoter uniquement les feuilles de l'arbre d'entrée et se base sur la connaissance préalable du schéma de l'arbre de sortie, tandis que l'autre s'appuie sur l'utilisation d'opérations d'édition d'arbres.

1.4.1 Annotation des feuilles par leur chemin

La première méthode d'annotation pour la transformation d'arbres XML que nous proposons possède deux caractéristiques principales :

– seules les feuilles de l'arbre (nœuds texte) d'entrée sont annotées.

[4] http://www.w3.org/2001/sw/

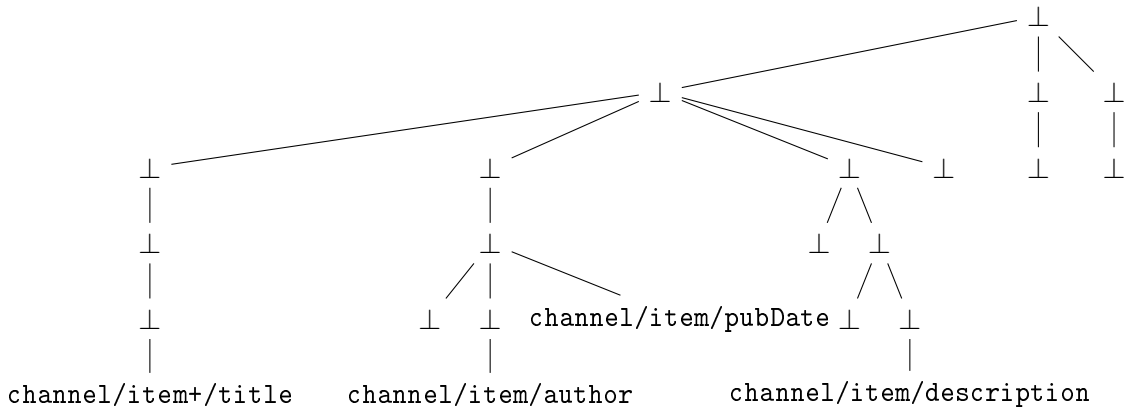

FIG. 1.14 – Annotation de type "chemin" de l'arbre de la figure 1.8 pour la tâche d'intégration de données au format RSS.

– la construction de l'alphabet des labels $\mathcal{Y}$ nécessite la connaissance préalable du schéma des arbres de sortie de la transformation.

Principe d'annotation

Le principe de cette méthode d'annotation consiste à identifier les nœuds texte (ou feuilles) de l'arbre d'entrée qui se retrouvent, à l'identique, dans l'arbre de sortie. Chacune de ces feuilles est annotée par la séquence des étiquettes des nœuds composant le chemin de la racine à cette feuille dans l'arbre de sortie, tandis que les feuilles qui n'apparaissent pas dans l'arbre de sortie sont annotées par le label $\perp$. Afin de constituer l'alphabet des labels nécessaire à la définition de la tâche d'annotation d'arbre, il faut donc connaître l'ensemble des chemins possibles, de la racine à une feuille, dans les arbres de sortie. C'est pourquoi cette méthode suppose que le schéma de sortie soit connu au préalable. Au moyen de ce schéma, il est possible de générer cet ensemble de chemins, et donc l'alphabet des labels. Toutefois, une approximation de cet alphabet peut tout de même être obtenue à l'aide d'un ensemble d'exemples d'arbres suivant la DTD de sortie. Un label est alors créé pour chaque chemin de la racine à une feuille de ces arbres. L'alphabet des labels ainsi créé n'est néanmoins pas assuré de correspondre à l'ensemble des chemins possibles dans le schéma de sortie.

Avec cette méthode d'annotation, si l'on reprend l'exemple de la transformation de l'arbre XHTML de la figure 1.8 en un arbre XML au format RSS (figure 1.9), on obtient l'arbre d'annotation de la figure 1.14. Sur cet arbre, on note par exemple que le nœud texte correspondant à l'auteur de l'information est annoté par `channel/item/author`, c'est-à-dire à son chemin dans l'arbre RSS de sortie. Le caractère `/` fait ici office de séparateur : `author` est un élément fils de `item`, lui-même étant un fils de `channel`. On remarque aussi que le titre de l'information est annoté par `channel/item+/title`. Le but de l'instruction `+` est décrit dans le paragraphe suivant, consacré à l'algorithme de construction de l'arbre de sortie à partir de l'arbre d'entrée annoté.

Construction de l'arbre XML de sortie

Algorithme 1 Algorithme de construction de l'arbre XML de sortie.

Entrée: Un couple $(\mathbf{x}, \mathbf{y})$ composé d'un arbre d'entrée et de son annotation.

```
1: L'arbre de sortie t est l'arbre vide.
2: pour chaque nœud x_i de x do
3:   si x_i est une feuille alors
4:     si y_i ≠ ⊥ alors
5:       elt ← racine(t)  # l'élément courant dans l'arbre de sortie est sa racine
6:       pour chaque etiq dans le chemin y_i do
7:         si etiq possède une instruction "+" ou elt ne possède pas de fils étiqueté par etiq alors
8:           Ajouter un élément new dont l'étiquette est etiq en dernier fils de elt
9:         sinon
10:          Soit new le dernier fils de elt étiqueté par etiq
11:        fin si
12:        elt ← new # new devient le nouvel élément courant
13:      fin pour
14:      Ajouter le contenu textuel de x_i en fils de elt
15:    fin si
16:  fin si
17: fin pour
```

Sortie: L'arbre de sortie t

Lorsqu'on dispose d'un arbre d'entrée dont les feuilles sont annotées par leur chemin dans l'arbre de sortie, l'algorithme 1 est utilisé pour construire l'arbre XML de sortie. Cet algorithme prend en entrée un arbre annoté de la façon décrite précédemment. L'arbre XML à générer est tout d'abord initialisé comme l'arbre vide. L'algorithme parcourt ensuite, dans l'ordre du document, tous les nœuds texte de l'arbre d'entrée qui ne sont pas annotées par $\perp$. Pour chacune de ces feuilles, l'algorithme parcourt, en partant de la racine, l'ensemble des étiquettes du chemin indiqué par l'annotation. Pour chaque étiquette *etiq*, l'algorithme fait un choix. Si aucun fils du nœud courant *elt* dans l'arbre de sortie en cours de génération n'est étiqueté par *etiq*, ou si l'instruction `+` est associée à *etiq*, alors un nouveau nœud étiqueté par *etiq* est ajouté en dernier fils de *elt*, ce nouveau nœud devenant le nouveau nœud courant. Sinon, le dernier fils de *elt* étiqueté par *etiq* devient le nouveau nœud courant. Une fois toutes les étiquettes du chemin traitées, le nœud texte en cours de traitement est ajouté comme fils du nœud courant dans l'arbre de sortie. Lorsque tous les nœuds texte de l'arbre d'entrée annoté ont été parcourus, l'algorithme renvoie en sortie l'arbre XML généré.

Nous illustrons maintenant cet algorithme sur l'exemple de transformation d'arbres au format RSS. L'algorithme prend donc en entrée l'arbre XHTML de la figure 1.8 et son annotation telle que représentée sur la figure 1.14. Les nœuds texte de cet arbre sont parcourus dans l'ordre du document. Le premier ("Give...") est annoté par `channel/item+/title`.

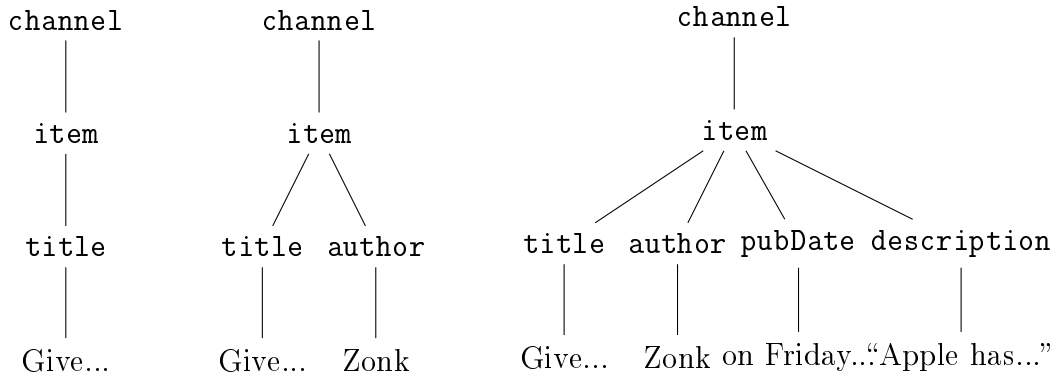

FIG. 1.15 – Différentes étapes de la génération de l'arbre de sortie avec l'algorithme 1.

L'algorithme parcourt alors les étiquettes des trois éléments correspondant au chemin représenté par ce label. Comme l'arbre de sortie est, au départ, vide, l'algorithme crée l'intégralité des éléments de ce chemin, et ajoute "Give..." en tant que feuille texte du nœud dont l'étiquette est `title`. Le premier arbre de la figure 1.15 est ainsi généré. La feuille texte suivante dans l'arbre d'entrée est celle dont le contenu textuel est "Posted By". Celle-ci est annotée par $\perp$ et n'est donc pas prise en compte. On considère ensuite la feuille dont le contenu est "Zonk" et qui est annotée par `channel/item/author`. L'algorithme parcourt alors les étiquettes des éléments du chemin correspondant à ce label. Dans un premier temps, à la racine du document, l'élément `channel` existe déjà. L'absence d'instruction `+` indique qu'il n'est pas nécessaire d'en créer un nouveau. L'algorithme se place donc dans ce nœud. De la même façon, ce nœud possède déjà un fils étiqueté par `item`, qui devient alors le nœud. Enfin, ce nœud ne possédant pas de fils étiqueté par `author`, celui-ci est créé, et la feuille texte "Zonk" est ajoutée. On obtient alors le deuxième arbre de la figure 1.15. L'algorithme continue de cette manière sur toutes les feuilles de l'arbre d'entrée et aboutit bien à l'arbre XML de sortie désiré (le troisième de la figure 1.15).

Critique

Cette méthode d'annotation pour la transformation présente un avantage majeur qui est de permettre, du point de vue de la structure interne, de modéliser des transformations dans lesquelles la structure de l'arbre d'entrée est radicalement différente de celle de l'arbre de sortie. En effet, la structure de sortie ne dépend ici en aucun cas de l'arbre d'entrée, et est intégralement représentée dans les labels.

Cette méthode possède toutefois quelques inconvénients. Le premier concerne l'ordre des feuilles. En effet, si la structure interne de l'arbre de sortie peut être radicalement différente de celle de l'arbre d'entrée, il n'en est pas de même pour les feuilles textes. En effet, l'ordre des feuilles texte dans l'arbre de sortie est contraint par leur ordre dans l'arbre d'entrée. Ainsi, sur l'exemple d'intégration de données précédent, il n'est pas possible de changer l'ordre des fils de l'élément `item` : par exemple, placer l'élément `description` avant `pubDate`. Toutefois, cette restriction peut en partie être contournée grâce à la connaissance de la DTD de sortie. Par exemple, si la DTD de RSS stipule que l'élément `description` doit précéder l'élément `pubDate`, la connaissance de cette DTD

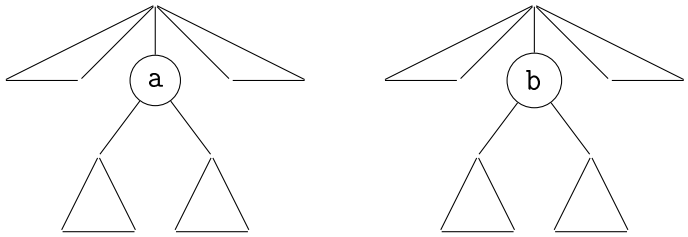

FIG. 1.16 – Renommer le noeud `a` en `b`.

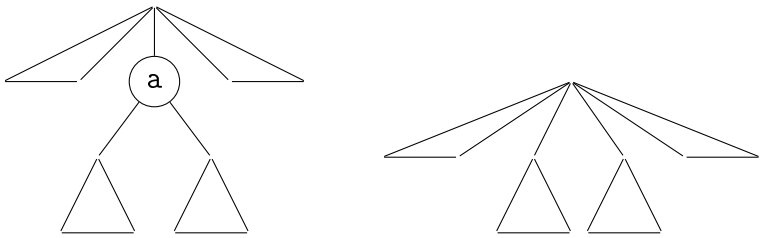

FIG. 1.17 – Supprimer le noeud `a`.

permet de réordonner ces deux éléments après la génération de l'arbre. L'ordre de deux éléments portant la même étiquette, par exemple deux éléments `item` d'un flux RSS, ne peut quant à lui pas être modifié de cette façon.

1.4.2 Annotation avec des opérations d'édition

Nous proposons maintenant une deuxième forme d'annotation d'arbres XML permettant de modéliser une tâche de transformation. Celle-ci s'appuie sur l'utilisation d'opérations d'édition d'arbres. En effet, nous proposons d'associer à chaque label avec lesquels seront annotés les arbres XML d'entrée une sémantique correspondant à diverses opérations d'édition d'arbres élémentaires. Parmi ces opérations, on trouve le renommage, l'insertion ou encore la suppression d'un nœud. Cette transformation, si elle ne nécessite pas forcément la connaissance complète du schéma cible de la transformation, nécessite tout de même de connaître l'ensemble des étiquettes de ce schéma, afin de connaître les étiquettes possibles des nœuds à insérer ou à renommer.

Opérations d'édition utilisées

Les différentes opérations d'édition d'arbres possibles que nous choisissons d'utiliser sont les suivantes :

- **renommer un nœud** : à chaque étiquette existant dans le schéma cible correspond un label de renommage de nœud. Par exemple, sur la figure 1.16, le nœud étiqueté par `a` est annoté par `b` et est donc renommé en `b` dans l'arbre de sortie.
- **supprimer un nœud** : quand un nœud est annoté par le label `delete`, il est supprimé, et ses fils deviennent alors les fils du nœud père, comme illustré sur la figure 1.17. Cette opération est naturellement interdite à la racine de l'arbre, celle-ci n'ayant par définition pas de père.

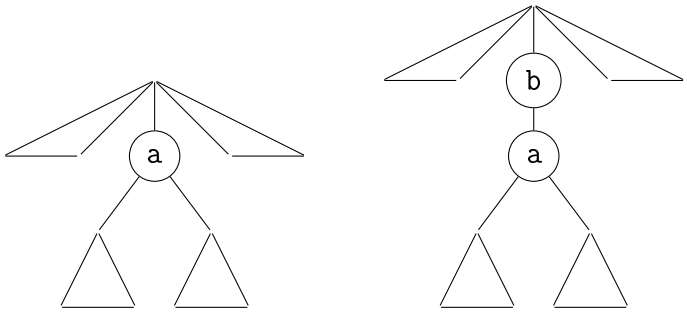

FIG. 1.18 – Insérer le noeud `b` en père du noeud `a`.

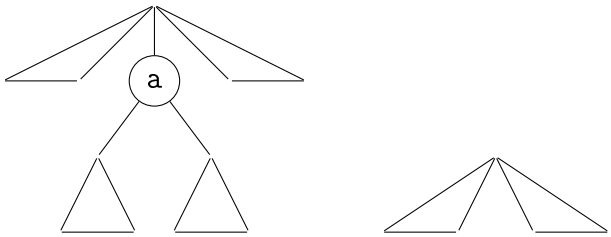

FIG. 1.19 – Supprimer le sous-arbre dont la racine est le noeud `a`.

- **insérer un nœud** : comme pour les opérations de renommage, il existe un label d'insertion de nœud pour chaque étiquette présente dans le schéma cible de la tâche de transformation. Lorsqu'un nœud `a` est annoté par le label d'insertion `insert_b`, un père étiqueté par `b` est ajouté au nœud. Ce nouveau nœud devient fils du père du nœud `a`, tandis que ce dernier reste inchangé. La figure 1.18 illustre cette opération d'édition.
- **supprimer un sous-arbre** : lorsqu'un nœud est annoté par le label `delST`, tout le sous-arbre enraciné à ce nœud est supprimé, comme le montre la figure 1.19. Cette opération est interdite à la racine de l'arbre, car il en résulterait un arbre vide.
- La dernière opération consiste à **laisser le nœud inchangé** quand celui-ci est annoté par le label $\perp$.

Avec les labels définis ci-dessus, on obtient un alphabet des labels dont la taille est $2 \times |T| + 3$, où $|T|$ est le nombre d'étiquettes dans le schéma cible T.

Génération de l'arbre XML de sortie

Un arbre XML annoté de cette manière définit une unique transformation de cet arbre en un autre arbre XML. Pour effectuer cette transformation, les opérations d'édition correspondant aux différents labels sont appliquées de manière descendante, de la racine de l'arbre vers les feuilles texte. Une simple feuille de transformation XSLT permet d'appliquer ces transformations. La transformation définie par une annotation est descendante et déterministe : elle garantit un arbre XML de sortie unique. La réciproque n'est toutefois pas vraie. En effet, plusieurs annotations différentes d'un même arbre XML peuvent produire un même arbre XML de sortie. Par exemple, avec l'alphabet des labels défini précédemment, la suppression d'un sous-arbre peut être exprimée de nombreuses façons

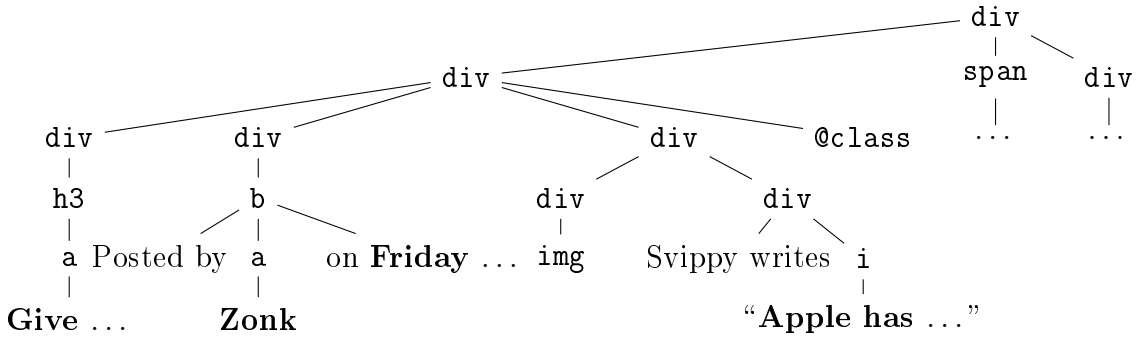

FIG. 1.20 – Exemple d'arbre XHTML pour la tâche de transformation RSS.

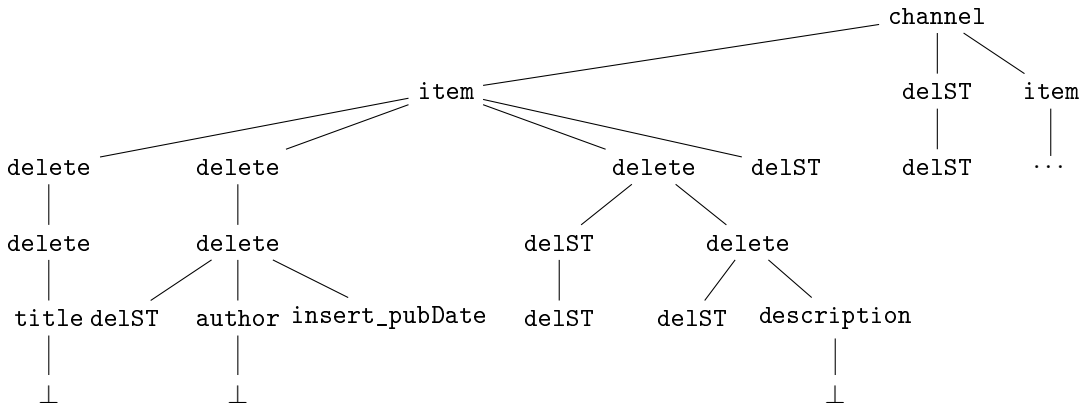

FIG. 1.21 – Annotation "opérations d'édition" de l'arbre XHTML de la figure 1.20.

différentes. En effet, les opérations d'édition étant appliquées de manière descendante, si la racine du sous-arbre est annotée par le label `delST`, l'ensemble du sous-arbre est supprimé, quels que soient les labels attribués aux autres nœuds de ce sous-arbre.

Nous illustrons maintenant cette technique de transformation sur l'exemple de la génération de flux RSS à partir d'une page Web. La figure 1.20 rappelle l'arbre XHTML de la page Web à partir de laquelle on souhaite générer un flux RSS. Avec l'ensemble des labels correspondant aux opérations d'édition définies précédemment, on obtient l'arbre d'annotation de la figure 1.21.

Cette annotation définit une transformation de l'arbre XHTML en un arbre RSS. Nous appliquons maintenant les opérations d'édition correspondant à cette annotation. La figure 1.22 montre diverses étapes de cette transformation. Les opérations étant appliquées de manière descendante, la première opération d'édition effectuée est donc celle de la racine. La racine `div`, en italique sur le premier arbre de la figure 1.22, est annotée par `channel`. Le nœud est donc renommé en `channel`, et l'algorithme d'application des opérations d'éditions passe donc au niveau suivant. Celui-ci comporte deux nœuds `div` annotés par `item` et un nœud `span` annoté par `delST`, en italique sur le deuxième arbre de la figure 1.22. Les deux premiers nœuds sont donc renommés en `item`, tandis que le sous-arbre complet dont la racine est le nœud `span` est supprimé. On obtient ainsi le troisième arbre intermédiaire de la figure 1.22. L'algorithme continue à appliquer les opérations de cette

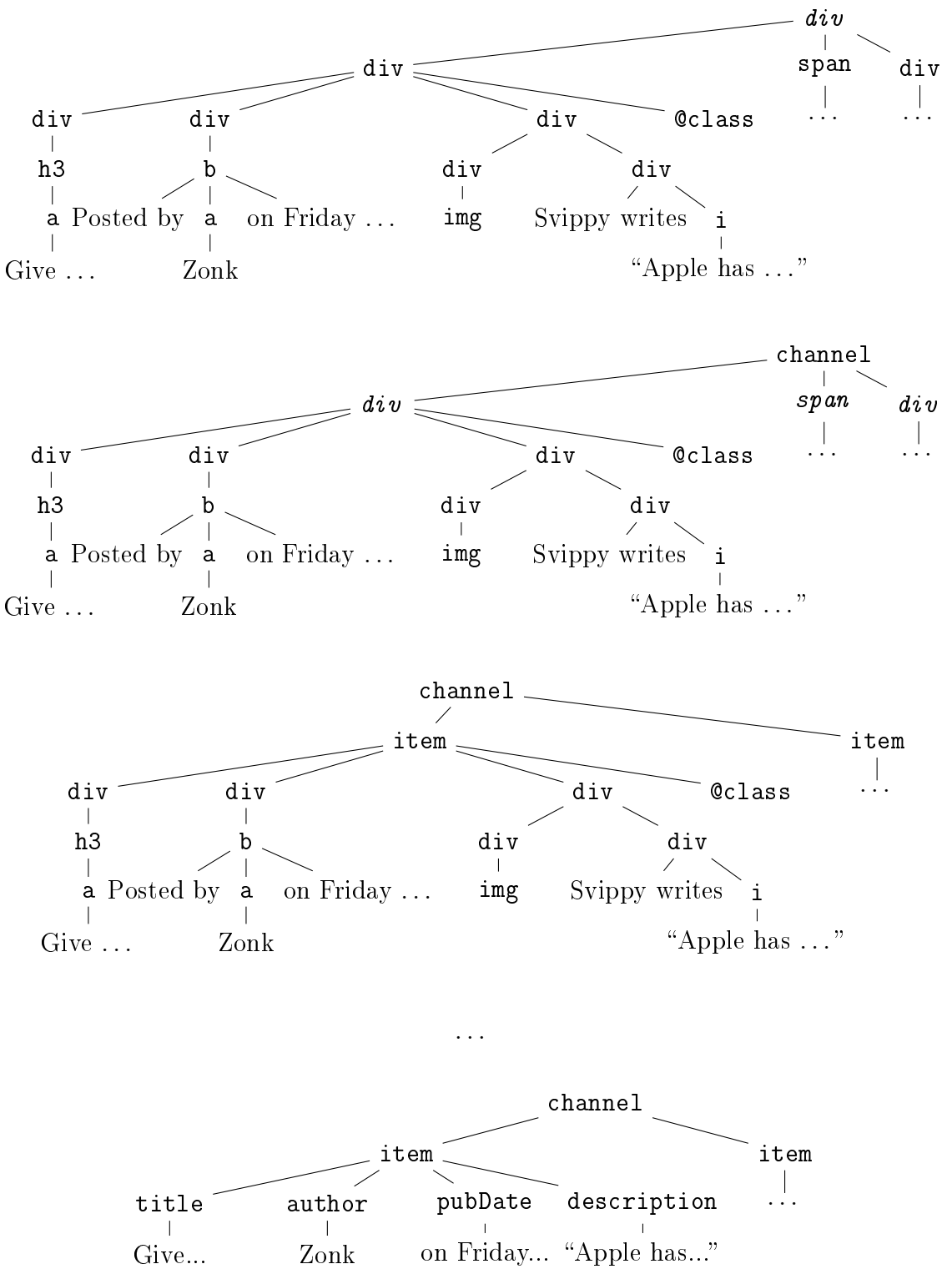

FIG. 1.22 – Transformation de l'arbre XHTML de la figure 1.20 vers l'arbre RSS par l'application des opérations d'édition affectées aux nœuds. Les nœuds sur lesquels sont appliqués les opérations sur signalés en italique.

manière, niveau par niveau, et on obtient ainsi l'arbre XML correspondant au flux RSS désiré.

Limitations

L'alphabet des labels que nous avons défini ici ne permet toutefois pas d'effectuer toutes les transformations possibles. Par exemple, il n'est pas possible avec cet alphabet des labels de renommer un nœud tout en insérant un nouveau nœud en tant que père de celui-ci. Cependant, plusieurs solutions permettent de résoudre ce problème. Une première solution serait d'ajouter un label pour chaque paire (`a`,`b`) d'étiquettes du schéma cible, en donnant à ce label la sémantique : renommer par `a` et insérer un père dont l'étiquette est `b`. Cette première solution engendre toutefois un nombre de labels quadratique dans le nombre d'étiquettes du schéma cible, ce qui n'est pas envisageable si le schéma cible comporte un trop grand nombre d'étiquettes. Une seconde solution consisterait à effectuer plusieurs annotations successives, une pour les renommages, une autre pour les insertions, permettant ainsi à un nœud d'être annoté par plusieurs labels.

Il n'est pas non plus possible de changer l'ordre des fils d'un nœud. Ce type d'opération ne paraît pas réalisable avec notre approche. En effet, les arbres XML étant des arbres d'arité non bornée, le nombre de combinaisons possibles est donc trop élevé et engendrerait un alphabet des labels de taille prohibitive.

Compte tenu des faibles possibilités de modification de la structure interne, en dehors des suppressions et des insertions de nœuds relativement limitées, cette méthode fait l'hypothèse d'une forte similarité entre la structure de l'arbre d'entrée et celle de l'arbre de sortie, contrairement à la précédente méthode d'annotation des feuilles par leur chemin. En effet, le squelette de l'arbre de sortie doit, à quelques insertions et suppressions près, être inclus dans celui de l'arbre d'entrée. Toutefois, cette seconde méthode d'annotation pour la transformation possède l'intérêt de définir des transformations simples, déterministes, tout en utilisant fortement l'ensemble de la structure de l'arbre d'entrée, contrairement à la méthode précédente qui elle consistait en l'annotation des seules feuilles. De plus, cette méthode est beaucoup plus souple et permet, par l'ajout de nouveaux labels correspondant à d'autres opérations d'édition, de modéliser des transformations plus complexes.

1.5 Conclusion

Nous avons, dans ce chapitre, introduit la problématique que nous abordons, c'est-à-dire la transformation d'arbres XML. Celle-ci a pour but d'obtenir, en sortie, des arbres XML suivant un schéma cible qui permet l'intégration et la réutilisation par d'autres applications des données contenues dans les arbres XML d'entrée. Bien qu'il existe de nombreuses façons de transformer des arbres XML, nous avons choisi de nous restreindre aux transformations qu'il est possible de modéliser par des annotations. Dans ce but, nous avons proposé deux méthodes d'annotation d'arbres XML définissant des transformations uniques de ces arbres. Nous allons maintenant, dans le chapitre suivant, nous intéresser à différentes méthodes d'annotations, et plus particulièrement aux techniques d'apprentissage automatique pour l'annotation et la transformation d'arbres XML.

Chapitre 2

Apprendre à annoter et à transformer

Il existe de nombreuses façons d'effectuer les tâches d'annotations et de transformations que nous venons de présenter. D'une part, celles-ci peuvent être effectuées manuellement ou à l'aide de programmes écrits spécialement pour une tâche donnée. Ainsi, des programmes de transformation d'arbres XML peuvent par exemple être écrits à l'aide du langage XSLT. Toutefois, de tels programmes sont spécifiques à une tâche précise, avec en entrée un ensemble d'arbres XML suivant un même schéma. L'adaptation de ces programmes à d'autres données et d'autres tâches nécessite donc une intervention humaine importante et de nombreuses heures de travail.

Pour remédier à cela, de nombreuses méthodes d'apprentissage automatique existent. Ces méthodes consistent à permettre à une machine d'apprendre à effectuer une tâche. Dans le cas présent, ces méthodes permettent d'apprendre à effectuer des tâches d'annotation d'arbre, à partir d'exemples d'arbre annotés. Dans un premier temps, nous présentons donc succinctement quelques méthodes d'apprentissage pour l'annotation et la transformation de séquences et d'arbres. De plus, celles-ci ne fournissant que très rarement des résultats absolument parfaits, il est nécessaire de les évaluer en comparant le résultat qu'elles proposent au résultat attendu. Nous présenterons donc quelques mesures d'évaluation de l'annotation et de la transformation qui nous servirons par la suite dans nos expériences.

2.1 Apprentissage supervisé pour l'annotation et la transformation

Nous nous intéressons donc dans un premier temps au problème de l'apprentissage automatique de programmes d'annotation et de transformation. Nous nous plaçons ici dans le cadre de l'apprentissage supervisé, c'est-à-dire l'apprentissage à partir d'exemples représentant des cas déjà traités. Dans le cas des tâches d'annotation et de transformation qui nous intéressent ici, ces exemples sont donc des couples $(\mathbf{x}, \mathbf{y})$ dont la première composante $\mathbf{x}$ est la séquence ou l'arbre d'entrée, tandis que la deuxième composante $\mathbf{y}$ est son annotation ou le résultat de la transformation.

2.1.1 Apprentissage pour l'annotation

Nous abordons dans un premier temps le cas des systèmes d'apprentissage pour l'annotation. Nous distinguons ici les techniques d'apprentissage pour l'annotation de séquences, qui sont nombreuses et qui ont été très largement étudiées, et celles pour l'annotation d'arbres, beaucoup plus rares.

Annotation de séquences

Dans le cas de l'annotation de séquences, le but est d'apprendre à associer une séquence de labels $\mathbf{y} = (y_1, \ldots, y_n)$ à une séquence $\mathbf{x} = (x_1, \ldots, x_n)$ correspondant à l'observation.

Une première méthode d'apprentissage pour l'annotation de séquences consiste à considérer une telle tâche d'annotation comme n tâches de classification multi-classes indépendantes où chaque élément x_i de l'observation est annoté par un label y_i prenant sa valeur dans l'alphabet des labels $\mathcal{Y}$. Ainsi, l'ensemble d'apprentissage est constitué non pas des couples de séquences $(\mathbf{x}, \mathbf{y})$ mais des couples (x_i, y_i). Pour cela, de nombreuses méthodes d'apprentissage pour la classification multi-classes existent, parmi lesquelles les méthodes de SVM multi-classes [Crammer and Singer, 2002, Hsu and Lin, 2002] ou encore les arbres de décision comme C4.5 [Quinlan, 1993] et C5 [Quinlan, 2004]. Cette façon de procéder ne tire toutefois pas partie de l'aspect structuré de l'annotation et des dépendances qui peuvent exister entre les différents labels de cette annotation.

Plusieurs méthodes d'apprentissage tiennent compte de cette structure en considérant l'annotation dans sa globalité, c'est-à-dire en n'annotant pas indépendamment chaque élément de la séquence. En effet, le choix d'un label à une position est susceptible d'affecter le choix d'un autre label dans la séquence. Parmi ces méthodes, une des premières à avoir fait ses preuves est celle des modèles de Markov cachés ou HMM (*Hidden Markov Models*), que nous décrivons plus en détail dans la section 3.2.2.0. Ceux-ci ont par exemple été appliqués avec succès à des tâches d'annotation *Part-of-Speech* [Lee et al., 2000] ou dans le cadre de l'extraction d'informations [Seymore et al., 1999].

Parmi les modèles plus récents permettant d'effectuer des tâches d'annotations, on trouve M^3N (*Maximum Margin Markov Networks*) [Taskar et al., 2003] ou encore SVM^{struct} [Tsochantaridis et al., 2005]. Ces deux méthodes sont des extensions des méthodes à noyaux, comme par exemple les SVM (*Support Vector Machines*). L'idée de base des SVM, dans le cas de la classification binaire, part du principe qu'il n'existe pas toujours un séparateur linéaire dans l'espace de description des données. Ainsi, les SVM se basent sur l'utilisation de fonctions appelées **noyaux** (*kernel*) qui permettent de reconsidérer le problème dans un espace de dimension supérieure dans lequel il existe un séparateur linéaire permettant d'effectuer correctement la tâche de classification. Ces méthodes, dans un premier temps créées dans le cadre de la classification binaire, sont, dans SVM^{struct} et M^3N, étendues au cadre plus général de la classification structurée multi-classes. Si elle ont permis d'obtenir de très bons résultats, à condition d'avoir défini un bon noyau, elles souffrent du problème de la pré-image. En effet, dans ces méthodes, lorsque l'image de la sortie dans l'espace défini par le noyau (*feature space*) a été trouvée, il est nécessaire de retrouver sa pré-image, c'est-à-dire son correspondant dans l'espace d'entrée. Cette procédure est potentiellement coûteuse en termes de complexité.

Un autre algorithme d'apprentissage permettant d'effectuer des tâches d'annotation de séquences est SEARN [Daumé III et al., 2006]. Celui-ci est un algorithme générique qui combine les algorithmes de recherche dans un espace (SEARch) et l'apprentissage (lEARN), qui ne se limite pas à l'annotation de séquences, mais permet d'effectuer de la prédiction de structures complexes (*Structured Output*). Le principe de ce méta-algorithme est de transformer un problème complexe en une succession de problèmes de classification multi-classes. Dans le cas de l'annotation de séquences, ces problèmes de classification consistent donc à trouver chaque label. Pour prédire ces labels, à chaque étape, l'algorithme peut s'appuyer à la fois sur l'ensemble de l'observation et sur tous les labels prédits aux étapes précédentes. Ainsi, SEARN s'affranchit de la propriété markovienne présente dans les HMMs. Toutefois, cet méthode reste fortement dépendante de l'ordre dans lequel est effectué l'annotation.

Enfin, le modèle des champs aléatoires conditionnels ou CRF (*Conditional Random Fields*) [Lafferty et al., 2001] permet lui aussi d'effectuer des tâches d'annotation de séquences. Ce modèle est décrit en détail dans la section 3.3. Tous ces modèles ont été récemment comparés sur une tâche d'annotation *Part-of-Speech* dans [Nguyen and Guo, 2007] et ont obtenu des performances comparables.

Annotation d'arbres

Les tâches d'annotation d'arbres ont pour leur part été beaucoup moins étudiées que l'annotation de séquences. En effet, on trouve les seuls exemples d'annotations d'arbres dans le cadre du traitement automatique du langage naturel, plus précisément pour la tâche d'annotation des rôles sémantiques d'une phrase. Dans ce domaine, comme dans le cas de l'annotation de séquences, une première solution consiste à effectuer une classification individuelle de chacun des nœuds de l'arbre à annoter. Dans ce cas, les mêmes méthodes d'apprentissage que pour les séquences, particulièrement les SVM multi-classes, peuvent être utilisées.

Toutefois, de la même façon que dans le cas de l'annotation de séquences, il peut être utile de considérer la tâche d'annotation d'un arbre dans sa globalité et non comme un ensemble de tâches de classification indépendantes. Dans le domaine de l'annotation des rôles sémantiques, une comparaison entre une méthode de classification indépendante des nœuds et une méthode tenant compte des dépendances entre les labels a été effectuée dans [Toutanova et al., 2005]. Cette comparaison a montré la supériorité de leur modèle utilisant ces dépendances. Celle-ci utilise un système de ranking (classement) afin de sélectionner la meilleure annotation possible parmi un ensemble d'annotations probables. On trouve aussi pour l'annotation d'arbres des méthodes d'apprentissage à base de noyaux telles que [Kashima and Tsuboi, 2004]. Celle-ci est adaptée aux tâches d'annotation sur tous types de structures, parmi lesquels les arbres. Toutefois, cette méthode n'est évaluée que dans le cas des séquences, son efficacité dans le cas des arbres reste donc à confirmer. De plus, de nombreuses techniques utilisées pour l'annotation de séquences et pour la prédiction de structures complexes présentées précédemment, telles que M^3N, SVM^{struct} ou encore SEARN, peuvent être utilisées pour l'annotation d'arbres.

À notre connaissance, aucune technique d'annotation d'arbres n'a à ce jour fait ses preuves dans le domaine précis du XML. Devant ce manque de techniques d'apprentissage

dédiées à cette tâche et à ce type de données, nous proposons, dans le chapitre 4, différents modèles de champs aléatoires conditionnels permettant d'apprendre à annoter des arbres XML.

2.1.2 Apprentissage pour la transformation d'arbres

Dans le domaine de l'apprentissage pour la transformation d'arbres XML, le *Schema Matching* (*cf.* section 1.2.2) occupe une place prépondérante. En effet, dans les années 2000, de nombreux systèmes d'apprentissage pour le *Schema Matching* ont vu le jour, parmi lesquels LSD [Doan et al., 2001], XMapper [Kurgan et al., 2002] ou encore iMap [Dhamankar et al., 2004]. Les deux premiers systèmes se limitent à trouver automatiquement des correspondances (*mappings*) 1-1 entre éléments des schémas XML d'entrée et de sortie. Toutefois, leurs approches diffèrent. D'un côté, le système LSD apprend ces correspondances à partir des schémas XML d'entrée et de sortie ainsi que d'exemples de documents dans ces schémas. L'apprentissage consiste alors en une première étape où plusieurs apprenants de base (par exemple Bayes Naïf) sont appris, suivie d'une seconde étape consistant en un méta-apprentissage qui vient combiner les prédictions des apprenants de base. XMapper, quant à lui, travaille directement à partir des arbres XML et ne nécessite pas de connaître les schémas d'entrée et de sortie. Son algorithme d'apprentissage se divise en deux parties. Dans un premier temps, une sélection d'attributs pour décrire les arbres XML est effectuée, puis, les correspondances 1-1 sont apprises en utilisant une mesure de distance entre les descriptions des arbres XML d'entrée et de sortie. Le système iMap permet lui d'apprendre des correspondances plus complexes que les correspondances 1-1, de la forme `adresse`=concat(`rue`,`ville`). Pour cela, le même principe que pour le système LSD est appliqué : plusieurs apprenants permettent de proposer différents types de correspondances (1-1, concaténation, *etc.*). Celles-ci sont évaluées et les meilleures sont conservées.

On trouve toutefois d'autres méthodes d'apprentissage pour la transformation d'arbres XML hors du domaine du *Schema Matching*. En effet, on trouve plusieurs travaux d'apprentissage de transformations d'arbres HTML orientés document en arbres XML. La méthode proposée par [Curran and Wong, 1999] s'appuie sur l'application d'une succession de règles de transformation, comme par exemple changer l'étiquette d'un nœud. Cette méthode consiste en un processus d'apprentissage itératif, où à chaque étape, plusieurs règles de transformations possibles sont proposées et évaluées par rapport au résultat attendu. Toutefois, aucun résultat expérimental n'est présenté pour venir illustrer les performances de cette méthode.

D'autres techniques de transformation d'arbres HTML ont aussi été proposées, parmi lesquelles [Chidlovskii and Fuselier, 2005]. Cette technique réduit le problème à une tâche d'annotation d'arbres : les documents HTML sont annotés sémantiquement par leur correspondant dans un schéma XML cible. Cette méthode se divise en deux étapes. Dans la première, les feuilles des arbres XML de sortie sont prédites à l'aide d'un classifieur à maximum d'entropie. Puis, la structure de l'arbre de sortie est retrouvée au moyen d'une grammaire hors-contexte probabiliste ou PCFG (*Probabilistic Context-Free Grammar*). Cette technique fait l'hypothèse que les feuilles sont dans le même ordre dans les arbres d'entrée et de sortie. Toutefois, la complexité de la deuxième étape est cubique

dans le nombre de feuilles : ce système ne peut donc s'appliquer à des arbres XML de grande taille.

Les travaux de [Gallinari et al., 2005] utilisent eux un modèle stochastique génératif pour transformer des arbres XML. Ce modèle calcule la probabilité que l'arbre XML de sortie ait été produit à partir de l'arbre d'entrée. Pour modéliser la probabilité d'un document, cette méthode sépare d'un côté la probabilité de la structure du document, c'est-à-dire l'ensemble des nœuds et de leurs étiquettes, et de l'autre côté la probabilité du contenu textuel du document sachant sa structure et le contenu textuel du document d'entrée. Si cette méthode à permis d'obtenir d'obtenir de bons résultats, elle possède toutefois le désavantage de ne tirer que très peu parti de la structure de l'arbre d'entrée pour générer le document XML de sortie. De plus, la génération de la sortie est relativement lente.

Enfin, un dernier algorithme d'apprentissage de transformations d'arbres est l'algorithme ISM (*Incremental Structure Mapping*) [Maes et al., 2007] s'inspire de techniques telles que LaSO [Daumé III and Marcu, 2005] ou SEARN [Daumé III et al., 2006]. Cette méthode parcourt dans l'ordre les feuilles de l'arbre d'entrée et génère l'arbre de sortie au fur et mesure. Pour chaque feuille, un certain nombre d'actions sont possibles (supprimer la feuille, l'insérer sous un nœud existant, créer de nouveaux nœuds pour l'insérer). Une fois toutes les feuilles parcourues, l'arbre de sortie est généré. Cette méthode offre de bons résultats et possède l'avantage d'être rapide en phase de test (génération des arbres XML de sortie). Toutefois, la phase d'apprentissage reste relativement lente.

2.2 Méthodes d'évaluation

Une fois une technique d'apprentissage automatique mise en œuvre, il est nécessaire de l'évaluer. En effet, une telle technique ne garantit jamais un résultat parfait, son évaluation permet donc non seulement de la comparer à d'autres, mais surtout de savoir quelle confiance lui accorder. Nous présentons donc tout d'abord le principe de l'évaluation d'un système d'apprentissage, avant de nous intéresser aux mesures d'évaluation pour l'annotation de séquences et d'arbres, mais aussi pour la transformation d'arbres XML.

2.2.1 Évaluation d'un système d'apprentissage supervisé

L'évaluation d'un système d'apprentissage supervisé consiste à mesurer la similarité entre le résultat produit par le système et le résultat correct, c'est-à-dire le résultat qu'obtiendrait un système parfait. Pour cela, il est donc nécessaire d'avoir à disposition ce résultat correct. Évaluer un système d'apprentissage supervisé nécessite donc de disposer d'un corpus composé de couples (entrée,sortie). Dans le cas de systèmes d'annotation d'arbres, par exemple, l'entrée est l'arbre à annoter, et la sortie est son annotation, tandis que dans le cas de la transformation, l'entrée est l'arbre à transformer, et la sortie est le résultat de cette transformation.

Une fois ce corpus à disposition, on le divise : une partie constitue l'ensemble d'apprentissage, tandis que l'autre partie constitue l'ensemble de test. Sur ce dernier est appliqué le système appris et les résultats obtenus sont alors comparés à ceux attendus au moyen

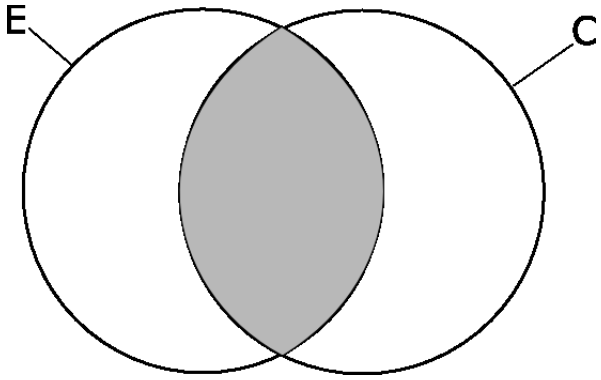

FIG. 2.1 – Schéma de l'évaluation. Le cercle de gauche représente l'ensemble E des données extraites. Le cercle de droite représente l'ensemble C des données à extraire. La partie grisée est l'intersection $E \cap C$, c'est-à-dire les données correctement extraites.

d'une mesure de similarité (*cf.* sections 2.2.2 et 2.2.3). Toutefois, lorsqu'on ne dispose pas d'un corpus suffisamment grand pour une évaluation efficace, il est possible d'utiliser la technique de la **validation croisée** à K blocs. Cette méthode consiste à partitionner le corpus en K blocs. Un bloc est retenu comme ensemble de test, tandis que les $K-1$ blocs restant sont utilisés comme ensemble d'apprentissage. Le processus est répété K fois pour chaque bloc. Chaque élément du corpus est donc apparu une fois et une seule dans un ensemble de test. Nous ne listons pas ici les propriétés de cette méthode et renvoyons aux travaux de [Kohavi, 1995] sur le sujet.

2.2.2 Évaluation de l'annotation

L'évaluation des systèmes d'apprentissage automatique pour l'annotation s'appuie sur des mesures d'évaluation classiques en extraction d'informations que sont la précision, le rappel et la F-mesure. Ces mesures considèrent l'existence de deux ensembles : l'ensemble E des données extraites par le système et l'ensemble C des données qui auraient dû être extraites (données Correctes). Le principe de ces mesures est d'évaluer la similarité entre ces deux ensembles. On représente ces deux ensembles sur la figure 2.1, avec à gauche l'ensemble E et à droite l'ensemble C. Le système d'apprentissage est parfait quand ces deux ensembles sont identiques, c'est-à-dire quand $E \cap C = E = C$. Dans le cas des systèmes d'apprentissage pour l'annotation, on peut donc évaluer la qualité du système pour chaque label l. L'ensemble E correspond donc à l'ensemble des nœuds ayant reçu le label l, tandis que C représente l'ensemble des nœuds qui auraient dû se voir attribuer ce label.

Dans un premier temps, la **précision** permet de mesurer la proportion de nœuds correctement annotés parmi les nœuds ayant été annotés avec le label l par le système :

$$P = \frac{|E \cap C|}{|E|} \tag{2.1}$$

Cette mesure donne donc une bonne idée de la confiance à accorder aux nœuds annotés par le label l : plus la précision d'un système est élevée, plus un nœud annoté par le label

l a de chances d'avoir été correctement annoté.

La deuxième mesure à notre disposition est le **rappel**. Celle-ci calcule la proportion de nœuds correctement annotés parmi les nœuds qu'il fallait annoter, pour un label l donné :

$$R = \frac{|E \cap C|}{|C|} \tag{2.2}$$

Cette mesure permet quant à elle d'évaluer la capacité du système à identifier un maximum de nœuds à annoter. Plus le rappel est élevé, moins il y a de risque que le système ait omis des nœuds.

En pratique, il est aisé de fournir un système d'apprentissage automatique qui fournit un rappel de 100%. En effet, pour obtenir un rappel de 100% pour un label l, il suffit d'annoter tous les nœuds de l'arbre ou de la séquence par ce label. Toutefois, une telle méthode fait significativement baisser la précision. À l'inverse, une méthode privilégiant la précision a tendance à faire diminuer le rappel. Le challenge de l'apprentissage automatique pour l'annotation est donc de fournir un système qui permet d'obtenir, sur la plupart des tâches, à la fois une bonne précision et un bon rappel. La **F-mesure** permet de mesurer ce compromis entre rappel et précision afin de donner une idée de la qualité générale du système :

$$F_\alpha = \frac{(1+\alpha)RP}{\alpha R + P} \tag{2.3}$$

Le paramètre α permet de privilégier le rappel ou la précision dans ce score. En pratique, la F_1-mesure ($\alpha = 1$), qui accorde un poids égal à la précision et au rappel, est la plus utilisée.

Afin d'évaluer un système d'apprentissage automatique sur une tâche d'annotation complète, il est nécessaire de considérer ses performances sur l'ensemble des labels de la tâche. Pour cela, on calcule la moyenne des performances obtenues par le système sur chacun des labels. Il existe toutefois plusieurs façons de calculer cette moyenne : nous en présentons ici deux. La première moyenne que l'on peut considérer est la **macro moyenne**. Cette méthode est une moyenne non pondérée, et consiste donc à donner la même importance à tous les labels de la tâche d'annotation. Pour une telle tâche, soient un alphabet de n labels $\mathcal{Y} = \{l_1, \ldots, l_n\}$, et $F_1(l_i)$ la F_1-mesure pour le label l_i. La macro moyenne de la F_1-mesure pour cette tâche s'exprime :

$$\text{macro}F_1 = \frac{1}{n}\sum_{i=1}^{n} F_1(l_i) \tag{2.4}$$

La seconde méthode de calcul de la moyenne des résultats est la **micro moyenne**. Cette seconde moyenne pondère le score de chaque label par le nombre de nœuds annotés par ce label. Soit une tâche d'annotation dont l'alphabet des labels de taille n est $\mathcal{Y} = \{l_1, \ldots, l_n\}$, et soient $F_1(l_i)$ la F_1-mesure pour le label l_i et $\tau(l_i)$ le nombre de nœuds annotés par le label l_i dans l'échantillon de test. La micro moyenne de la F_1-mesure pour cette tâche d'annotation s'exprime alors :

$$\text{micro}F_1 = \frac{1}{\sum_{i=1}^{n}\tau(l_i)}\sum_{i=1}^{n}\tau(l_i)F_1(l_i) \tag{2.5}$$

L'intérêt de cette deuxième méthode de moyenne est qu'elle accorde plus d'importance aux scores obtenus sur les labels les plus présents. Ainsi, si un système d'apprentissage obtient de mauvais résultats sur un ou plusieurs labels très peu présents, mais de bons résultats sur les autres labels, la micro moyenne restera assez élevé. Cette mesure donne donc un bon aperçu de la proportion de nœuds correctement annotés par le système. Toutefois, dans certaines applications, les labels les plus fréquents ne sont pas toujours les plus importants. Par exemple, parmi nos méthodes d'annotation pour la transformation, dans celle consistant à annoter les feuilles des arbres par leur chemin dans l'arbre de sortie, le label $\perp$ est toujours le plus présent, car affecté à tous les nœuds internes ainsi qu'aux feuilles non retenues dans la transformation. Dans un tel cas, l'utilisation de la micro moyenne pour résumer les scores peut entraîner un biais. Dans nos expériences, nous choisirons avec précaution la bonne moyenne, et donnerons aussi les résultats pour chaque label si nécessaire.

2.2.3 Évaluation de la transformation d'arbres XML

Nous décrivons maintenant les méthodes que nous allons utiliser pour évaluer la qualité d'une transformation d'arbres XML. Contrairement à l'évaluation en extraction d'informations ou en annotation, l'évaluation de la transformation d'arbres n'a été que très peu étudiée. Il existe toutefois des mesures de similarité d'arbres, basées sur la F-mesure, qui nous permettent d'évaluer une transformation en mesurant la similarité entre l'arbre transformé et l'arbre cible.

La première mesure de similarité d'arbres XML que nous utilisons s'appuie sur les chemins : pour chaque feuille de l'arbre XML résultant de la transformation, un couple composé du contenu textuel de cette feuille et des étiquettes du chemin de la racine à la feuille est créé. Un couple est considéré correct si le contenu texte et le chemin depuis la racine sont bons. L'ensemble de ces couples forment l'ensemble E des données extraites (*cf.* le schéma de la figure 2.1). L'ensemble C de ces couples composés à partir de l'arbre cible est aussi constitué. Il est alors possible de calculer précision, rappel et F-mesure à partir de ces deux ensembles. Cette première mesure donne un bon aperçu de la qualité de l'arbre mais ne tient pas compte de l'ordre dans lequel les feuilles apparaissent dans l'arbre de sortie, alors que les arbres XML sont des arbres partiellement ordonnés.

Pour tenir compte de l'ordre, une deuxième mesure basée sur les sous-arbres est proposée. Celle-ci consiste à considérer pour chaque nœud de l'arbre le sous-arbre enraciné à ce nœud. Deux sous-arbres sont égaux s'ils ont la même structure, les mêmes étiquettes aux nœuds et le même contenu textuel. Une fois encore, les ensembles E et C sont créés respectivement à partir des arbres transformés par le système et des arbres cibles, et la F-mesure est calculée sur ces ensembles de sous-arbres ainsi considérés.

Enfin, une dernière mesure vient combiner l'intérêt des deux précédentes. Celle-ci considère, pour chaque nœud de l'arbre XML, le couple composé du chemin depuis la racine et du sous-arbre enraciné à ce nœud. Un couple est considéré correct si le chemin depuis la racine et le sous-arbre sont corrects. Cette mesure est la plus stricte, car elle nécessite à la fois que chaque sous-arbre soit correctement situé dans l'arbre (grâce au test sur le chemin depuis la racine) et que ce sous-arbre soit strictement correct.

2.3 Conclusion

Dans ce chapitre, nous avons donc fait un tour d'horizon de différentes méthodes d'apprentissage automatique pour l'annotation et la transformation d'arbres. Celles-ci montrent que, si l'annotation de séquences est maintenant un domaine qui a été très largement étudié, c'est beaucoup moins le cas de l'annotation et de la transformation d'arbres XML. De plus, de nombreuses méthodes dans le domaine de la transformation d'arbres se limitent à des transformations très simples ou s'attaquent au problème plus général de la classification structurée (*Structured Output*), ne tenant ainsi pas compte des spécificités des arbres XML. C'est pourquoi nous proposons dans cette thèse une nouvelle méthode de transformation d'arbres XML basée sur l'annotation. Avant de présenter en détail notre modèle d'annotation d'arbres dans le chapitre 4, nous allons dans le chapitre suivant présenter les fondements sur lesquels il s'appuie.

Chapitre 3

Modèles graphiques pour l'annotation

Nous allons, dans ce chapitre, décrire la famille de modèles dont fait partie le modèle d'annotation d'arbres XML que nous présentons dans le chapitre suivant. Ces modèles, appelés modèles graphiques, permettent de représenter efficacement une distribution de probabilité. Nous présentons dans un premier temps les bases des modèles graphiques dans le cas général, en distinguant diverses familles et en décrivant les algorithmes d'inférence exacte et d'apprentissage pour ces modèles. Puis, dans une deuxième partie, nous nous concentrons sur les modèles graphiques pour l'annotation de séquences, avec plusieurs exemples, parmi lesquels les champs aléatoires conditionnels.

3.1 Les Modèles graphiques

3.1.1 Préliminaires

Nous introduisons dans un premier temps quelques notions nécessaires à la compréhension des modèles graphiques. Une **variable aléatoire** est une quantité dont les valeurs sont aléatoires, et à laquelle est associée une distribution de probabilité, de façon à ce qu'il soit possible de déterminer la probabilité que cette variable prenne une valeur donnée ou une valeur dans un intervalle donné. On parle de **variable aléatoire discrète** lorsque celle-ci prend ses valeurs dans un ensemble fini ou infini dénombrable. L'exemple de variable aléatoire discrète le plus simple est le résultat d'un lancer à pile ou face. L'ensemble dans lequel cette variable prend ses valeurs est $\{pile, face\}$. Par convention, nous notons toujours une variable aléatoire par une majuscule, par exemple X. Un **champ aléatoire** est un ensemble de variables aléatoires. On représente ce champ par une majuscule en gras : $\mathbf{X} = \{X_1, \ldots, X_n\}$.

On considère maintenant un ensemble de variables aléatoires discrètes $\{X_1, \ldots, X_n\}$. Chaque variable aléatoire X_i prend ses valeurs, notées x_i, dans l'ensemble $\mathcal{X}$. La probabilité pour que la variable aléatoire X_i prenne la valeur x_i est notée $p(X_i = x_i)$. Par souci de lisibilité, cette notation est souvent abrégée en $p(x_i)$. Avec un tel champ aléatoire se pose le problème de la représentation de la distribution de probabilité jointe $p(x_1, \ldots, x_n)$, principalement sa représentation en mémoire sur un ordinateur. La représentation "naturelle" d'une telle distribution consiste en un tableau de taille $|\mathcal{X}|^n$, représentant les

probabilités de toutes les valeurs que peut prendre le champ aléatoire $\mathbf{X}$. Ainsi, la taille de ce tableau est exponentielle dans le nombre de variables aléatoires. Une telle taille implique qu'avec seulement 50 variables aléatoires binaires, on dépasse déjà les capacités de mémoire des machines actuelles. Toutefois, une telle représentation de la distribution de probabilité jointe suppose que toutes les variables aléatoires du champ aléatoire $\mathbf{X}$ sont interdépendantes, c'est-à-dire que pour toutes variables X_i et X_j de $\mathbf{X}$, la valeur x_i prise par X_i influe sur la valeur x_j prise par X_j, et inversement. Toutefois, si la variable aléatoire X_1 ne dépend que de X_2, on a $p(x_1|x_2,\ldots,x_n) = p(x_1|x_2)$. La représentation de cette probabilité est alors de taille $|\mathcal{X}|^2$ au lieu de $|\mathcal{X}|^n$. Les **modèles graphiques** sont un moyen de représenter le plus efficacement possible de telles probabilités jointes portant sur de grands champs aléatoires en utilisant les indépendances entre les variables aléatoires. De plus, cette représentation simplifié permet aussi de réduire la complexité des algorithmes associés à ces modèles.

3.1.2 Représentation d'une probabilité avec un modèle graphique

Nous venons donc de voir qu'afin de représenter une distribution de probabilité jointe, il est nécessaire de définir des indépendances entre variables aléatoires. Nous allons maintenant voir comment les modèles graphiques permettent d'utiliser ces indépendances que nous définissons plus précisément maintenant.

Indépendances entre variables aléatoires

Dans la suite du document, si A représente un ensemble d'indices, nous notons $\mathbf{X}_A$ l'ensemble des variables aléatoires dont les indices sont dans l'ensemble A : $\mathbf{X}_A = \{X_i | i \in A\}$. De la même façon, $\mathbf{x}_A$ représente l'ensemble des valeurs prises par les variables aléatoires de l'ensemble $\mathbf{X}_A$.

On dit que deux ensembles de variables aléatoires $\mathbf{X}_A$ et $\mathbf{X}_B$ sont indépendants si on a

$$p(\mathbf{x}_A, \mathbf{x}_B) = p(\mathbf{x}_A)p(\mathbf{x}_B)$$

De la même façon, on dit que deux ensembles de variables aléatoires $\mathbf{X}_A$ et $\mathbf{X}_B$ sont conditionnellement indépendants sachant un ensemble de variables aléatoires $\mathbf{X}_c$ si :

$$p(\mathbf{x}_A, \mathbf{x}_B|\mathbf{x}_C) = p(\mathbf{x}_A|\mathbf{x}_C)p(\mathbf{x}_B|\mathbf{x}_C)$$

ou

$$p(\mathbf{x}_A, |\mathbf{x}_B, \mathbf{x}_C) = p(\mathbf{x}_A|\mathbf{x}_C)$$

La connaissance de telles indépendances permet donc de factoriser la distribution de probabilité jointe. Cette factorisation permet alors de représenter la distribution sous la forme de plusieurs tableaux de taille inférieure à $|\mathcal{X}|^n$, réduisant ainsi l'espace mémoire utilisé.

Représentation graphique des indépendances

Un modèle graphique utilise un graphe $\mathcal{G} = (V, E)$ afin de représenter les indépendances conditionnelles entre les variables aléatoires d'un champ aléatoire $\mathbf{X} = \{X_1, \ldots, X_n\}$. Un tel graphe est construit de la manière suivante :

- à chaque variable aléatoire X_i de $\mathbf{X}$ correspond exactement un sommet (ou nœud) $i \in V$ dans le graphe.
- les arêtes E du graphe représentent les indépendances conditionnelles entre les variables aléatoires. Ainsi, d'une manière simple, si deux variables aléatoires X_i et X_j sont indépendantes, il n'existe pas d'arête entre les nœuds i et j.

On appelle un tel graphe **graphe d'indépendances**. On distingue alors deux familles de modèles graphiques selon que le graphe d'indépendances est dirigé ou non. S'il est dirigé, on parle de **modèle graphique dirigé**. Dans le cas contraire, on parle de **modèle graphique non dirigé**.

3.1.3 Modèles graphiques dirigés

Nous présentons dans un premier temps les modèles graphiques dirigés. Ces modèles permettent de simplifier la représentation d'une distribution de probabilité jointe en introduisant des indépendances entre plusieurs variables aléatoires. Nous commençons par présenter deux exemples de modèles graphiques dirigés, avant de donner le calcul d'une probabilité dans un modèle graphique dirigé dans le cas général.

Quelques exemples

Un premier exemple de modèle dirigé consiste à représenter sous la forme de graphe d'indépendance la propriété suivante :

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1}) \tag{3.1}$$

Si l'on se limite à $n = 4$, cette propriété indique donc que :

$$p(x_1, x_2, x_3, x_4) = p(x_4|x_3, x_2, x_1)p(x_3|x_2, x_1)p(x_2|x_1)p(x_1)$$

La représentation graphique de cette distribution de probabilité est donnée sur la figure 3.1. Ce premier modèle ne réduit toutefois pas la taille de la représentation mémoire de cette distribution de probabilité.

On considère maintenant, comme second exemple de modèle dirigé, le cas des chaînes de Markov en temps discret. Le principe de ces chaînes est que la variable aléatoire à un temps t quelconque ne dépend que de la variable aléatoire au temps $t-1$. Ainsi, dans une chaîne de Markov de longueur n, pour tout triplet (i, j, k) tel que $j < k < i \leq n$, X_i et X_j sont conditionnellement indépendantes sachant X_k. En d'autres termes, la distribution de probabilité représentée par une chaîne de Markov respecte la propriété markovienne suivante :

$$\forall i \in [1, n-1],\ p(x_{i+1}|x_1, \ldots, x_i) = p(x_{i+1}|x_i) \tag{3.2}$$

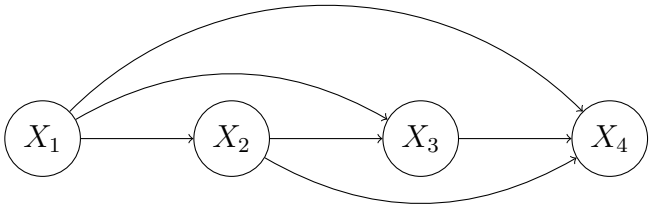

FIG. 3.1 – Graphe d'indépendances pour la propriété (3.1).

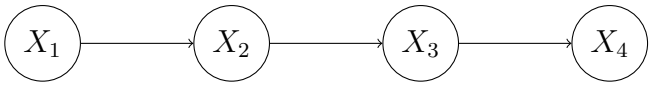

FIG. 3.2 – Graphe d'indépendances des chaînes de Markov en temps discret.

Le graphe d'indépendances d'une chaîne de Markov en temps discret de longueur $n = 4$ est représenté sur la figure 3.2.

La propriété markovienne (3.2), représentée par le graphe d'indépendances de la figure 3.2, et la propriété (3.1) impliquent donc que la distribution de probabilité jointe d'une telle chaîne de Markov se décompose ainsi :

$$p(x_1, x_2, x_3, x_4) = p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1) \tag{3.3}$$

La taille de la représentation de cette distribution est donc considérablement réduite. En effet, quelle que soit la longueur n de la chaîne de Markov considérée, la taille de la représentation passe de $|\mathcal{X}|^n$ à $|\mathcal{X}|^2$.

Probabilité dans un modèle dirigé

Nous nous intéressons maintenant aux indépendances conditionnelles dans les modèles graphiques dirigés en général. Pour cela, on définit tout d'abord la notion de **parent** dans un graphe dirigé.

Définition 3.1 *L'ensemble π_i des parents du nœud $i \in V$ d'un graphe $\mathcal{G} = (V, E)$ est l'ensemble des nœuds $j \in V$ tel qu'il existe une arête allant de j à i.*

$$\pi_i = \{j \in V | (j, i) \in E\}$$

Avec cette définition, une distribution de probabilité respectant un graphe $\mathcal{G}$ de taille n peut donc s'exprimer de la façon suivante :

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{\pi_i}) \tag{3.4}$$

Cette équation permet bien de retrouver les distributions de probabilité des équations (3.1) et (3.2).

3.1.4 Modèles graphiques non dirigés

Nous passons maintenant au cas des modèles graphiques non dirigés. Ceux-ci se définissent de la même façon que les modèles dirigés, à la différence que le graphe d'indépendances est ici un graphe non dirigé, c'est-à-dire un graphe $\mathcal{G} = (V, E)$ dans lequel l'existence d'une arête $(i, j) \in E$, avec $i, j \in V$, implique que l'arête $(j, i) \in E$ existe également. Ces modèles sont aussi appelés **champs aléatoires de Markov** (*Markov Random Fields*).

Indépendances conditionnelles

Comme exemple de modèle non dirigé, on considère la version non dirigée du graphe d'indépendances des chaînes de Markov de longueur $n = 4$. Ce graphe est représenté sur la figure 3.3.

FIG. 3.3 – Graphe d'indépendances non dirigé des chaînes de Markov.

Sur ce graphe, les variables aléatoires X_2 et X_4 sont conditionnellement indépendantes sachant X_3. D'une manière générale, on a la propriété suivante :

Définition 3.2 *Étant donné un graphe d'indépendances $\mathcal{G} = (V, E)$ et trois sous-ensembles de sommets distincts $A, B, C \subset V$, $\mathbf{X}_A$ et $\mathbf{X}_B$ sont conditionnellement indépendants sachant $\mathbf{X}_C$ si C sépare A de B dans le graphe $\mathcal{G}$.*

La notion de séparation est définie comme suit :

Définition 3.3 *Soient un graphe non dirigé $\mathcal{G} = (V, E)$ et trois sous-ensembles distincts de nœuds $A, B, C \subset V$, on dit que C sépare A de B si et seulement si tous les chemins d'un nœuds de A à un nœud de B passent par un nœud de C.*

Il en résulte que, dans un modèle non dirigé, deux variables aléatoires X_i et X_j dont les nœuds correspondants $i, j \in V$ ne sont pas directement connectés (il n'existe par d'arête $(i, j) \in E$) sont conditionnellement indépendantes sachant toutes les autres variables aléatoires.

Calcul de la probabilité

La factorisation de la probabilité jointe dans un modèle non dirigé n'est pas la même que dans le cas des modèles dirigés. Celle-ci s'appuie ici sur le fait que deux nœuds qui ne sont pas connectés correspondent à deux variables conditionnellement indépendantes. Cette propriété suppose que, dans la factorisation, ces deux variables doivent apparaître dans des facteurs différents. Ainsi, intuitivement, il est possible de factoriser cette probabilité en plusieurs facteurs portant sur les variables aléatoires correspondant à des sous-ensembles de nœuds connectés entre eux. On introduit donc la notion de clique :

Définition 3.4 *Une **clique** c est un sous-ensemble de nœuds ($c \subset V$) tel que pour toutes paires de nœuds $i, j \in c$, il existe une arête $(i, j) \in E$. On parle de **clique maximale** si celle-ci n'est pas contenue dans une autre clique du même graphe.*

Ainsi, sur l'exemple de graphe non dirigé de la figure 3.3, chaque singleton composé d'un nœud est une clique, et les cliques maximales sont les ensembles composés de deux nœuds consécutifs.

Le théorème de Hammersley-Clifford [Hammersley and Clifford, 1971] donne alors la factorisation de la distribution de probabilité dans un modèle graphique non dirigé.

Théorème 3.1 *Soit un graphe $\mathcal{G}$ dont l'ensemble des cliques est $\mathcal{C}$. La distribution de probabilité p d'un champ aléatoire de Markov (un modèle graphique non dirigé) est décomposable comme un produit de fonctions ψ_c définies sur les cliques c de l'ensemble $\mathcal{C}$ des cliques de $\mathcal{G}$:*

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c) \tag{3.5}$$

si et seulement si cette distribution de probabilités est strictement positive, c'est-à-dire si $p(\mathbf{x}) > 0$ pour tout $\mathbf{x}$ de $\mathcal{X}^n$. Z est un coefficient de normalisation assurant que $p(\mathbf{x})$ est bien une probabilité.

Les fonctions ψ_c de cette factorisation sont appelées **fonctions de potentiel**. Celles-ci sont des fonctions positives ou nulles, prenant en paramètre une réalisation $\mathbf{x}_c$ de l'ensemble de variables aléatoires $\mathbf{X}_c$. Le coefficient de normalisation Z s'exprime quant à lui de la manière suivante :

$$Z = \sum_{\mathbf{x} \in \mathcal{X}^n} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c) \tag{3.6}$$

Ainsi, sur l'exemple de modèle non dirigé de la figure 3.3, la distribution de probabilité jointe s'exprime comme suit :

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \psi_{\{1\}}(x_1) \psi_{\{2\}}(x_2) \psi_{\{3\}}(x_3) \psi_{\{4\}}(x_4) \\ \psi_{\{1,2\}}(x_1, x_2) \psi_{\{2,3\}}(x_2, x_3) \psi_{\{3,4\}}(x_3, x_4)$$

3.1.5 Algorithme d'inférence exacte pour les modèles graphiques

Étant donnée une distribution de probabilité définie par un modèle graphique, plusieurs problèmes d'**inférence** se posent :

- calculer la **probabilité marginale** $p(\mathbf{x}_A)$ pour un sous-ensemble de nœuds du graphe $A \subset V$.
- trouver la réalisation la plus probable $\hat{\mathbf{x}}$ du champ aléatoire $\mathbf{X}$, aussi appelée **maximum a posteriori** (MAP), c'est-à-dire trouver $\hat{\mathbf{x}} = \arg\max_{\mathbf{x} \in \mathcal{X}^n} p(\mathbf{x})$.
- trouver la réalisation la plus probable $\hat{\mathbf{x}}_A$ du sous-ensemble X_A du champ aléatoire $\mathbf{X}$ sachant la réalisation $\mathbf{x}_B$ du sous-ensemble $\mathbf{X}_B$, où A et B sont disjoints (souvent $A \cup B = V$) : $\hat{\mathbf{x}}_A = \arg\max_{\mathbf{x}_A \in \mathcal{X}^{|A|}} p(\mathbf{x}_A | \mathbf{x}_B)$

Ces problèmes, s'ils paraissent a priori différents, peuvent toutefois être traités selon le même principe. En effet, le calcul de la probabilité marginale $p(\mathbf{x}_A)$ peut s'exprimer de la manière suivante :

$$p(\mathbf{x}_A) = \sum_{\mathbf{x}_B \in \mathcal{X}^{|B|}} p(\mathbf{x}_A, \mathbf{x}_B) = \sum_{\mathbf{x}_B \in \mathcal{X}^{|B|}} p(\mathbf{x})$$

où B est le complémentaire de A dans V ($B = V \setminus A$). Or le maximum a posteriori est la réalisation $\hat{\mathbf{x}}$ qui maximise $p(\mathbf{x})$ pour toutes les valeurs de $\mathbf{x}$ possibles. Ainsi, dans ces définitions, la seule différence réside dans le remplacement de la fonction de somme par la fonction max. Enfin, le troisième problème combine les deux précédents. En effet, d'après la loi de Bayes, on a :

$$p(\mathbf{x}_A|\mathbf{x}_B) = \frac{p(\mathbf{x}_A, \mathbf{x}_B)}{p(\mathbf{x}_B)}$$

Ce calcul nécessite donc le calcul de deux probabilités marginales (une seule quand $A \cup B = V$). Le problème redevient alors une recherche du maximum a posteriori.

Afin de résoudre ces problèmes efficacement, des algorithmes d'inférence exacte existent. Ceux-ci sont des algorithmes de programmation dynamique mettant en œuvre un système de transmission de messages sur le graphe, ces messages correspondant à des résultats intermédiaires du calcul. Toutefois, ces algorithmes ne peuvent être appliqués que sur des graphes acycliques, ce qui est le cas des arbres. C'est pourquoi il est nécessaire, dans un premier temps, de construire une structure particulière, appelée arbre de jonction, sur laquelle les algorithmes pourront être appliqués. On appelle cet algorithme d'inférence exacte pour les modèles graphiques l'**algorithme de l'arbre de jonction** [Lauritzen and Spiegelhalter, 1990]. Nous commençons donc par décrire comment construire cette structure à partir du graphe d'indépendances, dirigé ou non. Nous présentons ensuite les algorithmes d'inférence exacte pour le calcul des probabilités marginales et du maximum a posteriori, avant de terminer sur la complexité de ces algorithmes.

Construction de l'arbre de jonction

Il est donc nécessaire, afin de pouvoir appliquer les algorithmes d'inférence décrits dans la suite, de construire l'arbre de jonction correspondant au graphe d'indépendances $\mathcal{G}$. Le schéma de la figure 3.4 représente les différentes étapes de la construction de cet arbre de jonction à partir du graphe d'indépendances. Ce schéma montre qu'une étape supplémentaire, appelé moralisation, est nécessaire dans le cas des graphes dirigés afin d'obtenir un graphe non dirigé. Ensuite, celui-ci doit être triangulé. Enfin, l'arbre de jonction peut être construit à partir de ce graphe triangulé. Nous décrivons maintenant ces trois étapes.

Dans le cas des modèles graphiques dirigés, la première étape consiste donc à moraliser le graphe d'indépendances. La **moralisation** consiste en la transformation d'un modèle graphique dirigé en un modèle graphique non dirigé. Moraliser un graphe se fait en deux étapes :

1. Pour chaque nœud n du graphe d'indépendances, on ajoute une arête non dirigée entre toutes les paires de parents de n qui ne sont pas déjà connectés. Par exemple,

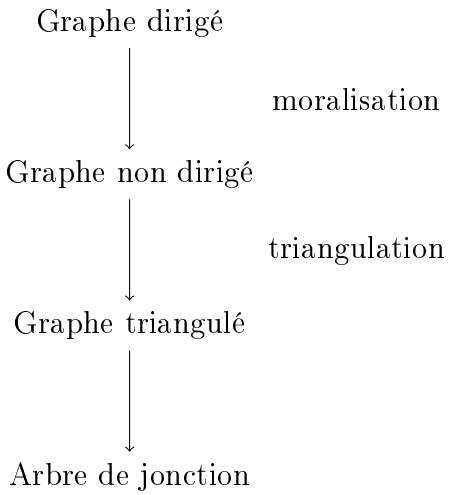

FIG. 3.4 – Schéma de la construction de l'arbre de jonction.

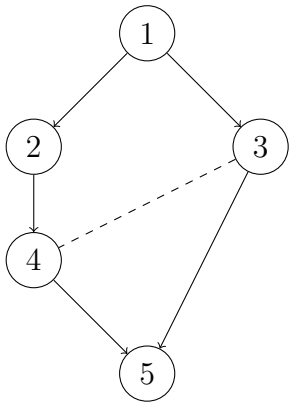

FIG. 3.5 – Graphe d'indépendances dirigé. L'arête en pointillés est ajoutée pour moraliser le graphe.

sur le graphe dirigé de la figure 3.5, on voit que les parents du nœud 5 ne sont pas connectés. Une arête non dirigée, représentée en pointillés, est donc ajoutée entre ces deux nœuds.

2. Toutes les arêtes dirigées du graphe sont transformées en des arêtes non dirigées.

Le concept sous-jacent de la moralisation est que, dans le graphe non dirigé qui en résulte, il est nécessaire que le nœud et l'ensemble de ses parents apparaissent dans une même clique. En effet, dans le modèle dirigé, le calcul de la probabilité jointe $p(\mathbf{x})$ fait intervenir les facteurs $p(x_i|\mathbf{x}_{\pi_i})$. Dans le modèle non dirigé correspondant, le calcul de la probabilité jointe fait donc intervenir des fonctions de potentiel de la forme $\psi_{\{i\}\cup\pi_i}(x_i, \mathbf{x}_{\pi_i})$.

Une fois que l'on dispose d'un graphe non dirigé tel que celui de la figure 3.6 (soit par moralisation, soit dans le cas d'un modèle graphique non dirigé), il est nécessaire, afin de pouvoir construire l'arbre de jonction, de **trianguler** le graphe. En effet, d'après [Lauritzen, 1996], on a la propriété suivante :

Propriété 3.1 *Un graphe $\mathcal{G}$ possède un arbre de jonction si et seulement si il est trian-*

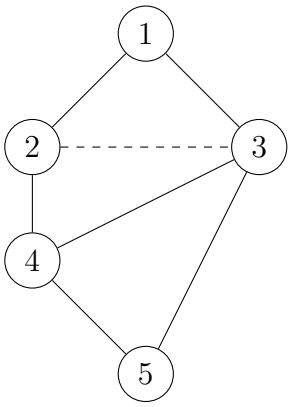

FIG. 3.6 – Graphe non triangulé (après moralisation). L'arête en pointillés est ajoutée pour trianguler le graphe.

gulé.

Afin de définir la notion de graphe triangulé, nous rappelons tout d'abord la définition d'un cycle dans un graphe non dirigé.

Définition 3.5 *Dans un graphe non dirigé $\mathcal{G} = (V, E)$, un* ***cycle*** *de longueur l est une séquence de noeuds $\mathcal{G}$ $(v_0, \ldots, v_l)$ distincts à l'exception de v_0 et v_l telle que $\forall 0 \leq i \leq l-1, (v_i, v_{i+1}) \in E$.*

On dit qu'un cycle possède une **corde** s'il existe une arête entre deux noeuds non consécutifs de ce cycle.

Définition 3.6 *Un graphe non dirigé $\mathcal{G}$ est* ***triangulé*** *si tous ses cycles de longueur $l \geq 4$ possèdent une corde.*

Le modèle des chaînes de Markov (figure 3.3), par exemple, puisqu'il ne contient pas de cycle, est déjà triangulé. Toutefois, dans le cas général, c'est-à-dire avec des graphes d'indépendances quelconques, il est nécessaire de passer par la phase de triangulation. Trianguler un graphe consiste à ajouter des cordes à tous les cycles de longueur supérieure ou égale à 4. Par exemple, considérons le graphe non dirigé de la figure 3.6. Dans ce graphe, le cycle $(1, 2, 4, 3, 1)$ de longueur 4 ne possède pas de corde. Afin de trianguler le graphe, il est donc nécessaire d'ajouter une corde entre les noeuds 2 et 3 comme dessiné en pointillés, ou entre les noeuds 1 et 4. Il existe en effet, pour un graphe non dirigé donné, plusieurs triangulations possibles. Par conséquent, un point à prendre en compte lors du choix d'une triangulation d'un graphe est la taille des cliques maximales. Sur l'exemple de la figure 3.6, la triangulation ne change pas cette taille. Toutefois, un des risques de la triangulation est de faire grandir significativement la taille des cliques maximales. Nous introduisons maintenant la notion de largeur d'arbre.

Définition 3.7 *La* ***largeur d'arbre*** *d'un graphe triangulé est la taille des cliques maximales moins* 1. *La largeur d'arbre d'un graphe non triangulé est la plus petite largeur d'arbre de toutes les triangulations possibles du graphe.*

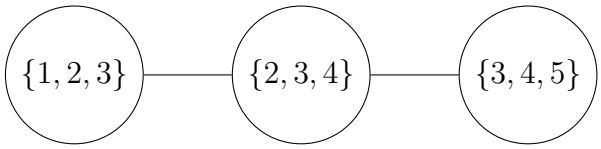

FIG. 3.7 – Arbre de jonction pour le graphe de la figure 3.6.

Si sur l'exemple de la figure 3.6, la triangulation du graphe est simple, trouver la **triangulation optimale** d'un graphe, c'est-à-dire la triangulation ayant la plus petite largeur d'arbre possible, s'avère être un problème NP-dur [Wen, 1991]. De plus, le choix de le triangulation est primordial, les algorithmes d'inférence exacte qui suivent étant exponentiel en la taille des cliques maximales du graphe triangulé.

Une fois que le graphe d'indépendances $\mathcal{G}$ sur lequel on travaille a été triangulé, on peut alors construire l'arbre de jonction qui lui correspond. L'arbre de jonction est une autre représentation du graphe, dans lequel les nœuds correspondent à des cliques du graphe.

Définition 3.8 *Un* ***arbre de jonction*** $\mathcal{T} = (V, E)$ *d'un graphe triangulé* $\mathcal{G}$ *est un arbre dans lequel les noeuds sont des cliques de* $\mathcal{G}$ *(*$V \subseteq \mathcal{C}$*) et dont les arêtes vérifient la* ***propriété d'intersection courante*** *: pour deux cliques* c *et* c' *de* $\mathcal{G}$*, les variables aléatoires contenues dans l'intersection de ces cliques sont contenues dans les cliques correspondant aux nœuds de l'arbre de jonction qui sont sur le chemin entre* c *et* c'*.*

Il existe, pour un graphe triangulé donné, plusieurs arbres de jonction possibles. Toutefois, le choix de cet arbre n'a pas d'incidence sur la complexité des algorithmes d'inférence. Dans le cas des chaînes de Markov, l'arbre de jonction du graphe d'indépendance est lui aussi une chaîne, mais dont les nœuds sont les cliques maximales, c'est-à-dire les paires de nœuds consécutifs dans le graphe. Un exemple d'arbre de jonction pour la figure 3.6 est représenté en figure 3.7. Cet arbre permet alors d'appliquer les algorithmes d'inférence exacte.

Calcul des probabilités marginales

Une fois l'arbre de jonction construit, afin de pouvoir exécuter les algorithmes d'inférence exacte sur cette structure, une première étape consiste à affecter à chaque nœud de l'arbre de jonction les fonctions de potentiel sur les cliques contenues dans ce nœud. Toutefois, certaines fonctions de potentiel peuvent en théorie être affectées à plusieurs noeuds, comme la fonction ψ_3 de l'exemple courant qui peut être affectée aux trois nœuds de l'arbre de jonction (figure 3.7). Sur l'exemple, nous avons choisi d'affecter les fonctions de potentiel au nœud le plus à gauche possible. À chaque nœud de l'arbre de jonction correspondant à une clique c, le produit des fonctions de potentiel affectées forme ainsi une fonction que nous notons ϕ_c. Sur l'exemple en figure 3.8, ces fonctions sont :

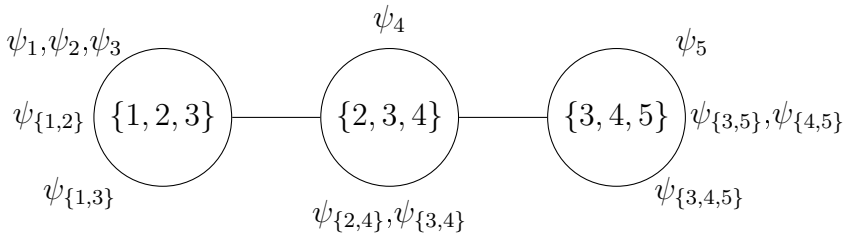

FIG. 3.8 – Arbre de jonction pour le graphe de la figure 3.6, décoré par les fonctions de potentiel.

$$\begin{aligned}
\phi_{\{1,2,3\}}(\{y_1,y_2,y_3\}) =& \psi_1(y_1)\psi_2(y_2)\psi_3(y_3)\psi_{\{1,2\}}(\{y_1,y_2\})\psi_{\{1,3\}}(\{y_1,y_3\}) \\
\phi_{\{2,3,4\}}(\{y_2,y_3,y_4\}) =& \psi_4(y_4)\psi_{\{2,4\}}(\{y_2,y_4\})\psi_{\{3,4\}}(\{y_3,y_4\}) \\
\phi_{\{3,4,5\}}(\{y_3,y_4,y_5\}) =& \psi_5(y_5)\psi_{\{3,5\}}(\{y_3,y_5\})\psi_{\{4,5\}}(\{y_4,y_5\})\psi_{\{3,4,5\}}(\{y_3,y_4,y_5\})
\end{aligned}$$

On définit maintenant sur cette structure des variables appelées **messages** qui permettent la mise en œuvre de l'algorithme de calcul des probabilités marginales couramment appelé algorithme *sum-product* (somme de produits). Le principe de cet algorithme est simple : des messages sont échangés entre les nœuds de l'arbre de jonction en suivant les arêtes et permettent de calculer récursivement les probabilités marginales, mais aussi le coefficient de normalisation $Z(\mathbf{x})$. Le message passant du nœud c' au nœud c dans l'arbre de jonction, que l'on note $\mu_{c'c}(\mathbf{x}_{c\cap c'})$, prend en paramètres les valeurs prises par les variables aléatoires appartenant à l'intersection des cliques c et c'. Si on note $\mathcal{N}(c)$ l'ensemble des nœuds voisins de c dans l'arbre de jonction, et $c' \setminus c$ l'ensemble des nœuds de c' n'appartenant pas à c, on peut alors exprimer un tel message de la façon suivante :

$$\mu_{c'c}(\mathbf{x}_{c\cap c'}) = \sum_{\mathbf{x}_{c'\setminus c}} \phi_{c'}(\mathbf{x}_{c'}) \prod_{c''\in\mathcal{N}(c')\setminus c} \mu_{c''c'}(\mathbf{x}_{c''\cap c'}) \tag{3.7}$$

Le principe de ces messages est de sommer le calcul des fonctions de potentiel de la clique c' sur tous les valeurs possibles des variables aléatoires de la clique c' qui ne sont pas dans la clique c et de calculer récursivement tous les messages passant des voisins c'' de c'. L'arbre de jonction étant, par définition, acyclique, l'algorithme a une fin. Celle-ci intervient quand, lors du calcul de $\mu_{c'c}(\mathbf{x}_{c\cap c'})$, le nœud c' n'a pas de voisin autre que c.

Avec ces messages, il est alors possible de calculer la probabilité marginale d'une clique c de la façon suivante :

$$p(\mathbf{x}_c) = \frac{1}{Z}\phi_c(\mathbf{x}_c) \prod_{c'\in\mathcal{N}(c)} \mu_{c'c}(\mathbf{x}_{c'\cap c}) \tag{3.8}$$

Sur l'exemple de la figure 3.7, si on appelle les trois cliques, de gauche à droite, c_1, c_2 et

c_3, le calcul de la probabilité marginale $p(x_{c_1})$ s'effectue donc de la façon suivante :

$$\begin{aligned} p(x_1, x_2, x_3) &= \frac{1}{Z}\phi_{c_1}(x_1, x_2, x_3)\mu_{c_2c_1}(x_2, x_3) \\ &= \frac{1}{Z}\phi_{c_1}(x_1, x_2, x_3) \sum_{x_4} \Big(\phi_{c_2}(x_2, x_3, x_4)\mu_{c_3c_2}(x_3, x_4)\Big) \\ &= \frac{1}{Z}\phi_{c_1}(x_1, x_2, x_3) \sum_{x_4} \Big(\phi_{c_2}(x_2, x_3, x_4) \sum_{x_5} \Big(\phi_{c_3}(x_3, x_4, x_5)\Big)\Big) \end{aligned}$$

De la même façon, l'algorithme *sum-product* permet aussi de calculer le coefficient de normalisation Z. Pour cela, il suffit de sa placer à n'importe quelle clique c dans l'arbre de jonction, et de sommer sur les probabilités marginales non normalisées :

$$\begin{aligned} \forall c \in \mathcal{C}, Z &= \sum_{\mathbf{x}_c} Z p(\mathbf{x}_c) \\ &= \sum_{\mathbf{x}_c} \phi_c(\mathbf{x}_c) \prod_{c' \in \mathcal{N}(c)} \mu_{c'c}(\mathbf{x}_{c' \cap c}) \end{aligned} \tag{3.9}$$

Calcul du maximum a posteriori

L'algorithme de programmation dynamique permettant le calcul du maximum a posteriori est quasiment identique à l'algorithme précédent. Pour ce nouvel algorithme, appelé algorithme *max-product*, il suffit de remplacer la fonction de somme présente dans la définition des messages par la fonction max. Les nouveaux messages s'expriment alors :

$$\delta_{c'c}(\mathbf{x}_{c \cap c'}) = \max_{\mathbf{x}_{c' \setminus c}} \phi_{c'}(\mathbf{x}_{c'}) \prod_{c'' \in \mathcal{N}(c') \setminus c} \delta_{c''c'}(\mathbf{x}_{c'' \cap c'}) \tag{3.10}$$

La probabilité du maximum a posteriori $\hat{\mathbf{x}}$ devient donc :

$$p(\hat{\mathbf{x}}) = \frac{1}{Z} \max_{\mathbf{x}_c} \phi_c(\mathbf{x}_c) \prod_{c' \in \mathcal{N}(c)} \delta_{c'c}(\mathbf{x}_{c' \cap c}) \tag{3.11}$$

Il suffit alors de mémoriser au fur et à mesure du calcul des variables δ les valeurs des variables aléatoires choisies dans la fonction max afin d'obtenir le maximum a posteriori $\hat{\mathbf{x}}$.

Complexité

Ces algorithmes d'inférence exacte sur l'arbre de jonction ont une complexité en temps et en espace de $\mathcal{O}(n \times |\mathcal{X}|^k)$, où n est la taille du graphe d'indépendances $\mathcal{G}$, c'est-à-dire le nombre de nœuds, $|\mathcal{X}|$ est le cardinal de $\mathcal{X}$, et k est la taille des cliques maximales de $\mathcal{G}$. En effet, les algorithmes nécessitent le calcul et la mémorisation de tous les messages, c'est-à-dire un message par nœud dans l'arbre de jonction. Or la taille de l'arbre de jonction est équivalente au nombre de cliques maximales dans le graphe d'indépendances triangulé qui

est au maximum la taille du graphe d'indépendances. De plus, ces messages doivent être calculés pour toutes les valeurs possibles prises par les variables aléatoires de ces cliques, soit un total de $|\mathcal{X}|^k$ possibilités.

Cette complexité montre les limitations des algorithmes d'inférence exacte dans les modèles graphiques et les précautions à prendre. En effet, avec un graphe d'indépendances trop complexe, c'est-à-dire avec de nombreuses dépendances, la taille k des cliques maximales peut vite être prohibitive. Il est donc raisonnable, pour utiliser ces algorithmes, de se limiter à des graphes relativement simples dans lesquels $k = 2$ ou $k = 3$. L'autre facteur à considérer est la taille du domaine $\mathcal{X}$. En effet, l'algorithme n'étant pas linéaire en $|\mathcal{X}|$, l'augmentation de la taille de $\mathcal{X}$ peut rendre les algorithmes d'inférence impraticables. Pour remédier à ces problèmes, des méthodes d'inférence approximatives ont été proposées, parmi lesquelles les méthodes dites variationnelles [Jordan et al., 1999] qui s'appuient sur l'utilisation d'une plus grande factorisation de la distribution de probabilité modélisée. Par exemple, si la triangulation du graphe d'indépendances augmente la taille des cliques maximales, ces méthodes utilisent la taille des cliques du graphe avant cette triangulation pour obtenir une approximation de la distribution de probabilité et ainsi réduire la complexité des algorithmes.

3.2 Cas de l'annotation de séquences

Nous nous intéressons maintenant aux cas des modèles graphiques pour l'annotation, et plus précisément ici pour l'annotation de séquences. En effet, les champs aléatoires représentés par ces modèles peuvent être divisés en deux sous-ensembles de variables aléatoires, eux-mêmes des champs aléatoires, l'un représentant une observation et l'autre son annotation. Ainsi, lorsque les paramètres du modèle graphique sont connus, il est possible, grâce aux algorithmes d'inférence, de trouver la réalisation la plus probable du champ aléatoire correspondant à l'annotation, connaissant l'observation. On distingue parmi les modèles graphiques pour l'annotation deux familles : les modèles génératifs et les modèles conditionnels ou discriminants. Ces deux familles se distinguent par le fait qu'elles modélisent soit la probabilité jointe des deux champs aléatoires, soit directement la probabilité conditionnelle de l'annotation sachant l'observation. Après une introduction sur l'annotation avec des modèles graphiques, nous présentons donc un exemple de modèle pour chacune des deux familles de modèles graphiques.

3.2.1 Annoter avec des modèles graphiques

Nous cherchons donc ici à résoudre des tâches d'annotation de séquences au moyen de modèles graphiques. Pour cela, les vecteurs $\mathbf{x} = (x_1, \ldots, x_n)$ et $\mathbf{y} = (y_1, \ldots, y_n)$ correspondant à la séquence observée et son annotation sont représentés par des champs aléatoires $\mathbf{X} = \{X_1, \ldots, X_n\}$ et $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$. Les variables aléatoires de ces deux champs prennent respectivement leurs valeurs dans les ensembles $\mathcal{X}$, le domaine de définition de l'observation, et $\mathcal{Y}$, l'alphabet des labels. Les vecteurs $\mathbf{x}$ et $\mathbf{y}$ deviennent donc des réalisations des champs aléatoires $\mathbf{X}$ et $\mathbf{Y}$. Étant donné un modèle graphique définissant une distribution de probabilité p, annoter une observation $\mathbf{x}$ revient donc à trouver

l'annotation $\hat{\mathbf{y}}$ maximisant la probabilité conditionnelle de $\mathbf{y}$ sachant l'observation $\mathbf{x}$:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} p(\mathbf{y}|\mathbf{x}) \tag{3.12}$$

Un modèle graphique pour l'annotation consiste donc en un graphe d'indépendances $\mathcal{G} = (V, E)$ tel qu'à chaque nœud $i \in V$ correspond une variable aléatoire dans un des deux champs aléatoires $\mathbf{X}$ et $\mathbf{Y}$. À la différence des modèles graphiques présentés dans la section 3.1, on a ici deux sous-ensembles de variables aléatoires $\mathbf{X}$ et $\mathbf{Y}$ bien distincts, prenant leurs valeurs dans deux domaines différents $\mathcal{X}$ et $\mathcal{Y}$. Toutefois, les principes fondamentaux de ces modèles restent les mêmes, et les algorithmes d'inférences présentés en 3.1.5 sont toujours applicables.

Il est cependant possible de distinguer deux familles de modèles graphiques permettant d'effectuer des tâches d'annotation. Celles-ci se distinguent par la façon dont elles représentent la probabilité conditionnelle de l'annotation sachant l'observation. D'une part, les modèles appelés modèles génératifs modélisent la probabilité jointe de l'annotation et de l'observation. Le calcul de la probabilité marginale de l'observation $p(\mathbf{x})$ permettant, grâce à la loi de Bayes, de calculer la probabilité conditionnelle :

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})}$$

D'autre part, les modèles conditionnels, aussi appelés modèles discriminants, modélisent quant à eux directement la probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$.

3.2.2 Modèles génératifs pour l'annotation

Les modèles de Markov cachés

Nous abordons dans un premier temps le cas des modèles génératifs. Pour cela, nous ne considérons ici que l'exemple du modèle le plus répandu dans cette famille : les modèles de Markov cachés ou HMMs (*Hidden Markov Models*). Ces modèles, à l'origine développés dans le cadre de la reconnaissance vocale [Rabiner and Juang, 1986], ont été très largement étudiés et ont obtenu de bons résultats dans diverses tâches d'annotation de séquences, à la fois en traitement du langage naturel et en extraction d'informations [Seymore et al., 1999, Freitag and McCallum, 2000, Skounakis et al., 2003]. Nous les présentons ici dans le cadre de tâches d'annotation en général. La figure 3.9 représente le graphe d'indépendances à la position i d'une séquence.

Sur les variables aléatoires Y_i correspondant aux labels, ce modèle possède la propriété markovienne suivante :

$$p(y_{i+1}|\mathbf{x}, y_1, \ldots, y_i) = p(y_{i+1}|y_i)$$

Cette propriété est, à la présence de $\mathbf{x}$ près, la même que dans les chaînes de Markov classiques (*cf.* section 3.1.3.0). Toutefois, le graphe d'indépendances montre qu'il existe aussi une forte hypothèse d'indépendance sur les variables aléatoires X_i correspondant à l'observation. Celle-ci s'exprime comme suit :

$$p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, x_n, \mathbf{y}) = p(x_i|y_i)$$

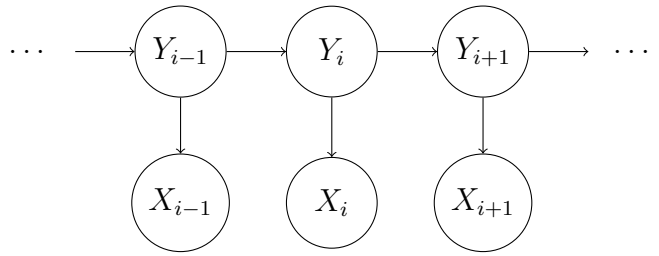

FIG. 3.9 – Graphe d'indépendances dirigé générique pour les modèles de Markov cachés.

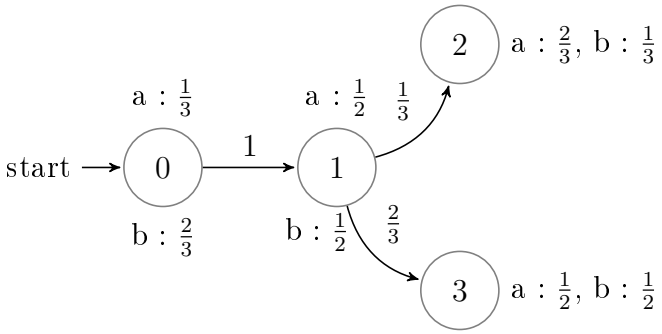

FIG. 3.10 – Vue automate stochastique d'un modèle de Markov caché.

En effet, dans ce modèle, l'observation x_i est "générée" par le label y_i choisi à la position i, d'où l'appellation de modèle génératif. Ces hypothèses d'indépendance nous permettent de représenter le calcul de la probabilité jointe de l'observation $\mathbf{x}$ et l'annotation $\mathbf{y}$:

$$p(\mathbf{x}, \mathbf{y}) = p(y_1)p(x_1|y_1) \prod_{i=2}^{n} p(y_i|y_{i-1})p(x_i|y_i) \tag{3.13}$$

Les modèles de Markov cachés peuvent aussi être vus comme des automates à états finis, dotés de transitions et d'émissions stochastiques. Dans cette vue, les états de l'automate représentent les labels, tandis que les émissions correspondent à l'observation. Ainsi, la probabilité de transition d'un état y_{i-1} à un état y_i correspond à la probabilité conditionnelle $p(y_i|y_{i-1})$ dans la formule de probabilité (3.13), tandis que la probabilité de l'émission d'un caractère x_i à l'état y_i correspond à $p(x_i|y_i)$. Enfin, la probabilité $p(y_1)$ correspond à la probabilité d'un état d'être état initial dans le HMM. La figure 3.10 propose la représentation sous forme d'automate d'un HMM. Celui-ci permet d'annoter des chaînes de longueur 3 définies sur le domaine $\mathcal{X} = \{a, b\}$, l'alphabet des labels correspondant à l'ensemble des états $\mathcal{Y} = \{0, 1, 2, 3\}$.

Si l'on veut, sur cet exemple, calculer la probabilité de la séquence "aaa", c'est-à-dire calculer $p(\mathbf{X} = aaa)$, on somme, pour toutes les annotations possibles $\mathbf{y}$ de $\mathbf{x} = aaa$, sur

les probabilités jointes $p(\mathbf{x}, \mathbf{y})$, de la façon suivante :

$$\begin{aligned} p(\mathbf{X} = aaa) &= \sum_{\mathbf{y} \in \mathcal{Y}^n} p(\mathbf{X} = aaa, \mathbf{Y} = \mathbf{y}) \\ &= p(\mathbf{X} = aaa, \mathbf{Y} = 012) + p(\mathbf{X} = aaa, \mathbf{Y} = 013) \\ &= \frac{1}{3} \times 1 \times \frac{1}{2} \times \frac{1}{3} \times \frac{2}{3} + \frac{1}{3} \times 1 \times \frac{1}{2} \times \frac{2}{3} \times \frac{1}{2} \\ &= \frac{1}{27} + \frac{1}{18} \\ &= \frac{5}{54} \end{aligned}$$

Sur cet exemple, étant donné qu'il n'y avait que deux annotations possibles pour la séquence "aaa", nous avons calculé les probabilités jointes directement. Toutefois, dans le cas général, l'algorithme *max-product* présenté dans la section 3.1.5.0 est utilisé. Son instanciation pour les HMMs est plus couramment connue sous le nom d'algorithme *forward-backward*. Cette algorithme, détaillé dans [Manning and Schütze, 1999], permet de calculer la probabilité d'une observation en $\mathcal{O}(n \times |\mathcal{Y}|^2)$.

Annoter avec un modèle de Markov caché

Si l'on revient à la définition de l'annotation avec un modèle graphique, on cherche, pour une observation $\mathbf{x}$ donnée, l'annotation $\hat{\mathbf{y}}$ maximisant la probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} p(\mathbf{y}|\mathbf{x})$$

Hors, d'après la loi de Bayes, on a :

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})}$$

De plus, comme l'annotation $\hat{\mathbf{y}}$ est recherchée pour une observation $\mathbf{x}$ donnée, $p(\mathbf{x})$ est une constante, et trouver la meilleure annotation de $\mathbf{x}$ revient donc à trouver l'annotation $\hat{\mathbf{y}}$ qui vérifie :

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} p(\mathbf{x}, \mathbf{y})$$

Pour trouver la meilleure annotation $\hat{\mathbf{y}}$, une méthode naïve consisterait à calculer cette probabilité pour toute les valeurs possibles de $\mathbf{Y}$. Toutefois, un tel algorithme est exponentiel dans la longueur de la séquence à annoter. Un algorithme de programmation dynamique appelé algorithme de Viterbi [Forney, 1973] permet donc de réaliser de façon efficace. Pour cela, on définit la variable $\delta_i(y_i)$. Celle-ci correspond à la probabilité de la meilleure annotation de la sous-séquence $x_1 \dots x_i$, où le label à la position i est fixé à $Y_i = y_i$.

$$\delta_i(y_i) = \max_{y_1 \dots y_i \in \mathcal{Y}^i} p(X_1 \dots X_i = x_1 \dots x_i, Y_1 \dots Y_i = y_1 \dots y_i) \tag{3.14}$$

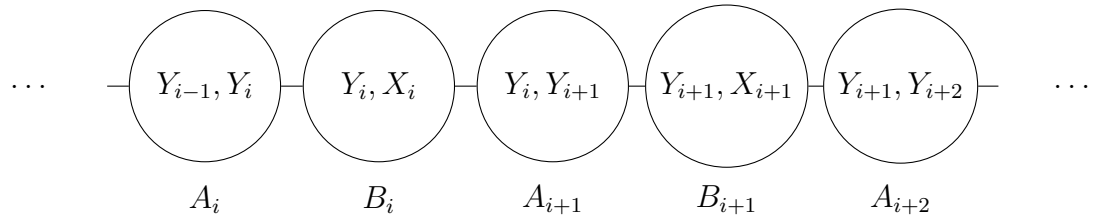

FIG. 3.11 – Arbre de jonction pour les modèles de Markov cachés

A l'aide de cette variable, il est alors possible de calculer en temps quadratique (en la taille de l'alphabet des labels $\mathcal{Y}$) la probabilité de la séquence d'états (l'annotation) la plus probable $\hat{\mathbf{y}}$ pour une observation $\mathbf{x}$ donnée. Cette variable δ est définie récursivement comme suit :

$$\begin{aligned}
\text{Initialisation} &: \delta_1(y_1) = p(y_1)p(x_1|y_1) \\
\text{Récursion} &: \delta_{i+1}(y_{i+1}) = p(x_{i+1}|y_{i+1}) \max_{y_i \in \mathcal{Y}} \delta_i(y_i) p(y_{i+1}|y_i) \\
\text{Fin} &: p(\mathbf{x}, \hat{\mathbf{y}}) = \max_{y_n \in \mathcal{Y}} \delta_n(y_n)
\end{aligned}$$

Lors du déroulement de l'algorithme, à chaque étape i, on mémorise le y_i choisi dans la fonction de max. La séquence des y_i choisis constitue alors la meilleure annotation $\hat{\mathbf{y}}$. L'algorithme de Viterbi permet d'effectuer une tâche d'annotation avec une complexité de $\mathcal{O}(n \times |\mathcal{Y}|^2)$.

On peut noter que l'algorithme de Viterbi que nous venons de décrire est un cas particulier de l'algorithme *max-product* sur l'arbre de jonction correspondant au graphe d'indépendances des HMMs. En effet, l'arbre de jonction pour ce graphe, représenté sur la figure 3.11, est une séquence dont les nœuds correspondent aux cliques de la forme $\{Y_{i-1}, Y_i\}$, que nous notons A_i, et de la forme $\{X_i, Y_i\}$, notées B_i. En reprenant la définition des variables δ de l'algorithme *max-product* (*cf.* équation (3.10)), on a donc un premier type de message :

$$\begin{aligned}
\delta_{B_{i+1}A_{i+2}}(y_{i+1}) &= \max_{x_{i+1}} p(x_{i+1}|y_{i+1}) \delta_{A_{i+1}B_{i+1}}(y_{i+1}) \\
&= p(x_{i+1}|y_{i+1}) \delta_{A_{i+1}B_{i+1}}(y_{i+1})
\end{aligned}$$

En effet, comme nous voulons trouver le maximum a posteriori pour une observation $\mathbf{x}$ donnée, x_{i+1} est fixe, et $\max_{x_{i+1}} p(x_{i+1}|y_{i+1}) = p(x_{i+1}|y_{i+1})$. On a ensuite un second type de message :

$$\delta_{A_{i+1}B_{i+1}}(y_{i+1}) = \max_{y_i} p(y_{i+1}|y_i) \delta_{B_i A_{i+1}}(y_i)$$

De ces deux expressions des messages, on obtient alors :

$$\begin{aligned}
\delta_{B_{i+1}A_{i+2}}(y_{i+1}) &= p(x_{i+1}|y_{i+1}) \delta_{A_{i+1}B_{i+1}}(y_{i+1}) \\
&= p(x_{i+1}|y_{i+1}) \max_{y_i} p(y_{i+1}|y_i) \delta_{B_i A_{i+1}}(y_i) \\
&= \delta_{i+1}(y_{i+1})
\end{aligned}$$

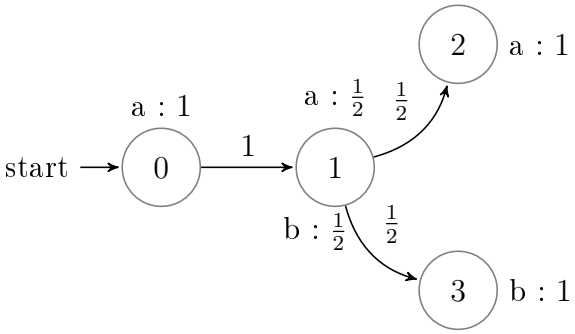

FIG. 3.12 – HMM appris à partir de l'échantillon d'apprentissage $S = \{(\texttt{aaa}, 012), (\texttt{abb}, 013)\}$.

Apprentissage dans les modèles de Markov cachés

Nous venons de voir que les HMMs permettent de calculer efficacement à la fois la probabilité d'une observation et la meilleure annotation possible pour une observation donnée. Toutefois, ces calculs nécessitent de connaître les probabilités d'émission des observations, de transition d'un état à un autre, et la probabilité pour un état d'être initial. Ces probabilités forment les paramètres du système. Il est donc raisonnable de vouloir, à partir d'exemples, apprendre ces paramètres. Dans ce but, deux cas sont possibles :

- on dispose, comme exemples d'apprentissage, uniquement des observations $\mathbf{x}$. Les annotations (ou les séquences d'états) $\mathbf{y}$ ne sont pas connues.
- on dispose de couples $(\mathbf{x}, \mathbf{y})$ correspondant à une observation et son annotation.

Le premier cas est le plus courant lorsque les HMMs sont utilisés dans le but de calculer la probabilité $p(\mathbf{x})$ d'une observation. L'algorithme de *Baum-Welch*, un algorithme de type EM (*Expectation-Maximization*), permet d'estimer les paramètres du HMM afin de maximiser la vraisemblance de l'échantillon d'apprentissage fourni. Nous ne décrivons pas cet algorithme ici et renvoyons à la lecture de [Manning and Schütze, 1999].

Le second cas paraît plus adapté au cas des tâches d'annotation, même s'il est en pratique peu étudié. C'est aussi le cas le plus simple. En effet, il suffit dans ce cas de compter. Pour trouver la probabilité de la transition $p(2|1)$, on compte la proportion de transitions de 1 vers 2 dans l'échantillon d'apprentissage parmi toutes les transitions partant de 1. De même, pour les émissions de type $p(a|1)$, il suffit de calculer la proportion de a émis à l'état 1 parmi toutes les observations émises à l'état 1. Ainsi, si on considère la structure du HMM de la figure 3.10, et que l'on a comme échantillon d'apprentissage $S = \{(\texttt{aaa}, 012), (\texttt{abb}, 013)\}$, on obtient le HMM de la figure 3.12. On se place par exemple à l'état 1. On considère tout d'abord les émissions. Dans le premier exemple, un `a` est émis, tandis qu'un `b` est émis dans le second exemple. On a donc $p(\texttt{a}|1) = \frac{1}{2}$ et $p(\texttt{b}|1) = \frac{1}{2}$. Pour les transitions partant de 1, on a, dans l'échantillon d'apprentissage, une transition vers 2 et une autre vers 3. Ainsi, $p(2|1) = \frac{1}{2}$ et $p(3|1) = \frac{1}{2}$.

Toutefois, avec une méthode d'apprentissage aussi simpliste, les risques de surapprentissage (*overfitting*) sont grands. En effet, dans l'exemple qui précède, aucun exemple ne commençait par un `b`. Ainsi, avec le HMM appris, les probabilités de toutes les séquences commençant par ce caractère sont nulles. Pour éviter ce problème, des techniques de lis-

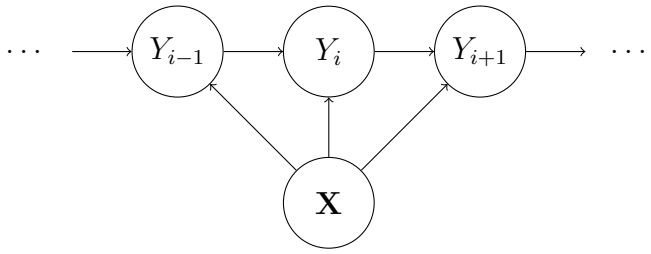

FIG. 3.13 – Graphe d'indépendances d'un MEMM à la position i.

sage, comme par exemple le lissage de Laplace, sont employées. Ces méthodes permettent essentiellement d'éviter d'apprendre des probabilités nulles.

Conclusion

Les modèles génératifs sont donc un bon outil pour effectuer des tâches d'annotations. En effet, si l'on considère le cas des modèles de Markov cachés, ils possèdent des algorithmes efficaces de programmation dynamique pour l'annotation et pour l'apprentissage. De plus, ils ont obtenu de bons résultats sur diverses tâches d'annotation dans le domaine du traitement du langage naturel, et plus particulièrement sur des tâches comme l'annotation *Part-Of-Speech* [Lee et al., 2000] ou la segmentation de textes [Sjölander, 2003], mais aussi en extraction d'informations [Freitag and McCallum, 1999].

Toutefois, dans le cadre de certaines tâches d'annotation, la connaissance du domaine $\mathcal{X}$ de définition des observations peut être limitée. Or, les modèles génératifs nécessitent de modéliser la probabilité $p(\mathbf{x})$. Nous présentons maintenant les modèles conditionnels qui ne nécessitent pas de modéliser l'observation.

3.2.3 Modèles conditionnels pour l'annotation

Les modèles conditionnels tiennent leur nom du fait qu'ils modélisent directement la probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$ de l'annotation sachant l'observation. Ils évitent ainsi de devoir modéliser l'observation et sa probabilité $p(\mathbf{x})$. Nous présentons ici ces modèles conditionnels avec l'exemple des modèles de Markov à maximum d'entropie.

Les modèles de Markov à maximum d'entropie

Les modèles de Markov à maximum d'entropie ou *Maximum Entropy Markov Models* (MEMMs) [McCallum et al., 2000] ont été définis dans le cadre de tâches d'extraction d'informations. Ceux-ci sont un modèle graphique dirigé et peuvent être considérés comme l'équivalent conditionnel des modèles de Markov cachés. Le graphe d'indépendances à la position i pour les MEMMs est représenté sur la figure 3.13.

Les MEMMs restent des modèles de Markov en cela qu'ils respectent eux aussi, sur les variables aléatoires Y_i correspondant à l'annotation uniquement, la propriété markovienne :

$$p(y_{i+1}|y_1, \ldots, \mathbf{y}_i) = p(y_{i+1}|y_i)$$

Toutefois, contrairement aux HMMs, le modèle des MEMMs ne "génère" pas l'observation. À l'inverse, chaque variable aléatoire Y_{i+1} dépend non seulement de la variable aléatoire Y_i la précédant, mais aussi de l'ensemble du champ aléatoire $\mathbf{X}$. Ainsi, on a :

$$p(y_{i+1}|\mathbf{x}, y_1, \ldots, \mathbf{y}_i) = p(y_{i+1}|\mathbf{x}, y_i)$$

Avec ces dépendances, on obtient, pour ce modèle, l'expression de la distribution de probabilité jointe suivante :

$$p(\mathbf{y}, \mathbf{x}) = p(\mathbf{x})p(y_1)\prod_{i=2}^{n} p(y_i|y_{i-1}, \mathbf{x})$$

Enfin, en divisant par $p(\mathbf{x})$ et en appliquant la loi de Bayes, on obtient la distribution de probabilité conditionnelle (3.15), justifiant l'appellation de modèle conditionnel des MEMMs.

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p(y_i|y_{i-1}, \mathbf{x}) \tag{3.15}$$

Dans le but de simplifier les notations, on suppose que $p(y_1) = p(y_1|y_0)$.

Afin de représenter les probabilités conditionnelles de la forme $p(y|y', \mathbf{x})$, les MEMMs utilisent des **fonctions de transition** $p_{y'}(y, \mathbf{x})$. Une telle fonction est définie pour chaque label de l'alphabet des labels $\mathcal{Y}$. Ces fonctions sont calculées à l'aide de modèles exponentiels. Ainsi, chaque fonction de transition s'exprime de la façon suivante :

$$p_{y'}(y, \mathbf{x}) = \frac{1}{Z(\mathbf{x}, y')} \exp\Big(\sum_{k=1}^{K} \lambda_k f_k(y', y, \mathbf{x})\Big) \tag{3.16}$$

Ces fonctions s'expriment donc sous la forme de l'exponentielle d'une combinaison linéaire de K fonctions f_k et de coefficients λ_k. Cette exponentielle est normalisée par la fonction $Z(\mathbf{x}, y')$. Celle-ci garantit que la fonction de transition représente bien une distribution de probabilité conditionnelle sachant $\mathbf{x}$ et y'. Ce coefficient s'exprime comme suit :

$$Z(\mathbf{x}, y') = \sum_{y_i \in \mathcal{Y}_n} \exp\Big(\sum_{k=1}^{K} \lambda_k f_k(y', y, \mathbf{x})\Big)$$

Les fonctions f_k utilisées dans ce calcul sont appelées **fonctions de caractéristiques** (*feature functions*) et sont des fonctions prenant en paramètres les labels y' et y correspondant aux valeurs de deux variables aléatoires consécutives dans le graphe d'indépendances, ainsi que l'ensemble de l'observation $\mathbf{x}$. Ces fonctions sont des fonctions à valeurs réelles. Toutefois, elles sont souvent utilisées comme des fonctions binaires valant 1 si un test est vrai, 0 sinon. Les poids $\Lambda = \{\lambda_1, \ldots, \lambda_K\}$ associés à ces fonctions forment les paramètres du MEMM. On considère l'exemple de la tâche de transformation d'arbres XML (*cf.* section 1.2.1) à partir de la page Web de la figure 3.14. Si l'on traite ce problème comme une tâche d'annotation de séquences, la fonction de caractéristiques suivante est utilisée pour annoter l'auteur de l'information.

$$f(y_i, y_{i+1}, \mathbf{x}) = \begin{cases} 1 & \text{si } y_i = 0,\ y_{i+1} = \texttt{author} \\ & \text{et si } x_{i-1}..x_i = \text{"Posted by" et } x_{i+1} \text{ est souligné} \\ 0 & \text{sinon} \end{cases}$$

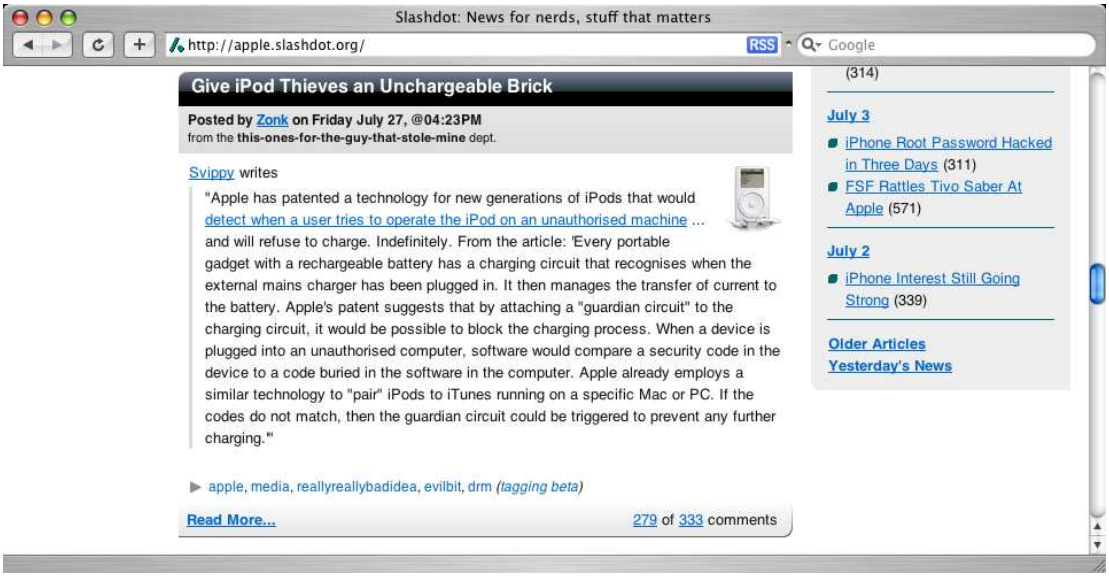

FIG. 3.14 – Rappel de la page Web issue du site Slashdot.

De plus, la définition même de ces fonctions implique que des dépendances longue distance sur l'observation peuvent être exprimées, ce qui n'était pas possible dans les modèles de Markov cachés où le label choisi à la position i ne dépend que du label précédent et du mot à la position i. En effet, les fonctions de caractéristiques prennent en paramètre l'intégralité de l'observation, ce qui a pour effet que n'importe quelle partie de l'observation peut influencer le choix d'un label. Par exemple, l'exemple précédent de fonction de caractéristiques ne porte pas uniquement sur x_i, mais aussi sur x_{i-1} et x_{i+1}.

Annoter avec un MEMM

L'annotation avec un MEMM est très similaire à l'annotation avec un HMM. Ainsi, on cherche l'annotation $\hat{\mathbf{y}}$ qui maximise la probabilité $p(\mathbf{y}|\mathbf{x})$, étant supposés connus les ensembles de fonctions de caractéristiques $F = \{f_1, \ldots, f_K\}$ de chaque fonction de transition et leurs poids Λ. La connaissance de ces paramètres permet de calculer les fonctions de transition $p_{y'}(y, \mathbf{x})$. Pour résoudre cette tâche d'annotation, l'algorithme de Viterbi, décrit rapidement dans la section 3.2.2.0, est adapté au cas des MEMMs de la façon suivante :

$$\begin{aligned} \text{Initialisation} &: \delta_1(y_1) = p(y_1) \\ \text{Récursion} &: \delta_{i+1}(y_{i+1}) = \max_{y_i \in \mathcal{Y}} \delta_i(y_i) p_{y_i}(y_{i+1}, \mathbf{x}) \\ \text{Fin} &: p(\hat{\mathbf{y}}|\mathbf{x}) = \max_{y_n \in \mathcal{Y}} \delta_n(y_n) \end{aligned}$$

Dans cette adaptation de l'algorithme de Viterbi, seule la formule de récursion est modifiée. Avec cet algorithme, il est possible, comme pour les HMMs, de résoudre les tâches d'annotation avec une complexité de $\mathcal{O}(n \times |\mathcal{Y}|^2)$.

Apprentissage dans les MEMM

Nous abordons maintenant l'apprentissage dans un MEMM. Pour rappel, la distribution de probabilité conditionnelle définie par un MEMM s'exprime de la manière suivante :

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p(y_i|y_{i-1}, \mathbf{x})$$

Ainsi, apprendre un MEMM consiste à appendre les probabilités $p(y_i|y_{i-1}, \mathbf{x})$. Or, nous l'avons vu précédemment, celles-ci s'expriment sous la forme de modèles exponentiels, dont les paramètres sont le vecteur Λ des poids des fonctions de caractéristiques. Ainsi, apprendre un MEMM revient donc, étant donnés en entrée un échantillon d'apprentissage composé de couples $(\mathbf{x}, \mathbf{y})$ et les ensembles de fonctions de caractéristiques pour chaque fonction de transition, à trouver les poids Λ de ces fonctions de caractéristiques qui maximisent un critère d'optimisation. Le critère le plus couramment utilisé est le maximum de vraisemblance, décrit plus en détail dans la section 3.3.2.0. L'apprentissage d'un MEMM consiste alors en une succession de $|\mathcal{Y}|$ apprentissages : un par fonction de transition $p_{y'}$.

Dans [McCallum et al., 2000], une approche est aussi proposée permettant d'apprendre les paramètres Λ avec en entrée un échantillon d'apprentissage composé uniquement des observations, les annotations n'étant pas connues. Cette méthode est une adaptation de l'algorithme de Baum-Welch utilisé dans les HMMs.

Problème du biais de label

Toutefois, les MEMMs, s'ils ont obtenus de bons résultats sur des tâches d'annotation pour l'extraction d'informations, souffrent du **problème du biais de label**. Ce problème est lié à la définition même des MEMMs et à la normalisation locale effectuée dans ce modèle. En effet, en sommant sur tous les labels possibles à la position i d'une séquence, on obtient :

$$\begin{aligned}\sum_{y_i} p(y_i|y_{i-1}, x_1, \ldots, x_i) &= \sum_{y_i} p(y_i|y_{i-1}, x_i).p(y_{i-1}|x_1, \ldots, x_{i-1}) \\ &= 1.p(y_{i-1}|x_1, \ldots, x_{i-1}) \\ &= p(y_{i-1}|x_1, \ldots, x_{i-1})\end{aligned}$$

Cela montre que la masse de probabilité reçue par y_{i-1} est complètement transmise à y_i, quelle que soit la valeur de x_i. Ce dernier ne sert qu'à déterminer la valeur prise par y_i. Dans le cas où aucun y_i n'est compatible avec x_i, l'ensemble de la masse de probabilité est tout de même transmise à y_i.

Pour illustrer ce problème, considérons l'exemple d'une tâche devant distinguer deux chaînes de caractères "rib" et "rob". L'alphabet des labels est $\mathcal{Y} = \{1, 2\}$, et l'annotation de "rib" est "111", celle de "rob" est "222". Considérons le cas de l'annotation de la chaîne "rob". On observe d'abord $X_1 = r$. La probabilité de Y_1 est équitablement répartie en $p(Y_1 = 1|X_1 = r) = p(Y_1 = 2|X_1 = r) = 0.5$. Le premier label est donc choisi au hasard. Supposons que le label 1 a été choisi. Lors de l'observation de $X_2 = o$ et du choix de la valeur de Y_2, la transition de $Y_1 = 1 \rightarrow Y_2 = 2$ a une probabilité nulle, et le label 1

est donc choisi, indépendamment de l'observation de $X_2 = o$. Enfin, le même phénomène se produit lors du choix de la valeur de Y_3. Le MEMM a donc annoté "rob" par "111" sans même tenir compte de l'observation $X_2 = o$. Il aurait tout aussi bien pu annoter correctement cette chaîne par "222" avec la même probabilité.

Dans le but d'éviter ce problème, [Lafferty et al., 2001] ont défini en 2001 un nouveau modèle pour l'annotation de séquences appelé les **champs aléatoires conditionnels**. Ceux-ci ne souffrent pas du problème du biais de label car ils ne mettent pas en œuvre une normalisation locale comme dans les MEMMs, mais une normalisation globale sur l'ensemble de l'annotation.

3.3 Les champs aléatoires conditionnels pour les séquences

Nous présentons donc maintenant le modèle des champs aléatoires conditionnels pour les séquences, c'est-à-dire celui adapté aux tâches d'annotations de séquences, introduit par [Lafferty et al., 2001].

3.3.1 Modèle

Les champs aléatoires conditionnels ou CRFs (*Conditional Random Fields*) sont, tout comme les MEMMs, un modèle conditionnel, définissant ainsi une probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$. Toutefois, ils sont un modèle graphique non dirigé et possèdent donc un graphe d'indépendances non dirigé représentant les hypothèses d'indépendances faites sur les variables aléatoires $X_1 \dots X_n$ et $Y_1 \dots Y_n$ correspondant respectivement à l'observation et à l'annotation. Le graphe d'indépendances couramment utilisé dans le cas des séquences est celui des chaînes du premier ordre représenté sur la figure 3.15. Ce graphe est le même que celui des MEMM, mais en non dirigé. L'hypothèse d'indépendance effectuée dans ce cas est la suivante :

$$\forall 1 \leq i \leq n,\ p(Y_i|\mathbf{X}, Y_j, j \neq i) = p(Y_i|\mathbf{X}, Y_{i-1}, Y_{i+1})$$

Le choix d'un label à la position i dépend donc des labels aux positions $i-1$ et $i+1$ ainsi que de l'ensemble de l'observation $\mathbf{X}$. Des graphes d'indépendances plus complexes, telles que les chaînes du second ordre (une dépendance existe aussi entre les variables aléatoires Y_{i-1} et Y_{i+1}), auraient aussi pu être utilisés, mais cela entraînerait une augmentation de la complexité des algorithmes d'inférence (*cf.* section 3.1.5.0).

D'après l'équation (3.5), la distribution de probabilité jointe dans ce modèle s'exprime donc comme un produit de fonctions de potentiel sur les cliques de ce graphe :

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z}\psi_{\mathbf{X}}(\mathbf{X})\psi_{\{Y_{i-1}\}}(y_{i-1})\psi_{\{Y_i\}}(y_i)\dots\psi_{\{Y_{i-1},Y_i,\mathbf{X}\}}(y_{i-1}, y_i, \mathbf{x})$$

où Z est le coefficient de normalisation. Pour simplifier les notations, on note la fonction de potentiel sur la clique $\{Y_i\}$ $\psi_i(y_i)$. De même, sur la clique $\{Y_i, Y_j\}$, on choisit la notation $\psi_{i,j}$.

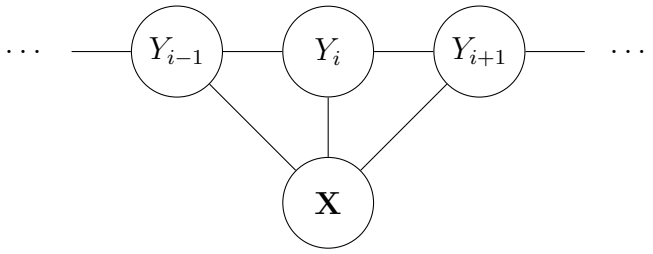

FIG. 3.15 – Graphe d'indépendances d'un CRF pour les séquences à la position i.

De la même façon que dans les MEMMs, on veut ici modéliser une probabilité conditionnelle. En ne tenant pas compte des fonctions de potentiel exclusivement sur $\mathbf{X}$, et en supposant que les fonctions de potentiel de la forme $\psi_{\{Y_i\}}(y_i)$ sont un cas particulier de $\psi_{\{Y_i,\mathbf{X}\}}(y_i, \mathbf{x})$, on obtient l'expression de la distribution de probabilité conditionnelle suivante :

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} \prod_{i=1}^{n} \psi_i(y_i, \mathbf{x}) \prod_{i=2}^{n} \psi_{i-1,i}(y_{i-1}, y_i, \mathbf{x})$$

Dans la suite du document, lorsque que nous parlerons du graphe d'indépendances d'un CRF, nous ferons uniquement référence au graphe sur les variables aléatoires Y_i correspondant à l'annotation. L'observation $\mathbf{X}$ sera ainsi toujours implicitement un nœud connecté à toutes les variables aléatoires Y_i. De même, l'ensemble $\mathcal{C}$ des cliques fera référence à l'ensemble des cliques de ce graphe sans $\mathbf{X}$.

Sur le même principe que les fonctions de transition des MEMMs, les fonctions de potentiel ψ_i et $\psi_{i-1,i}$ s'expriment, dans les champs aléatoires conditionnels, sous la forme de modèles exponentiels. Ainsi, on a :

$$\psi_i(y_i, \mathbf{x}) = \exp(\sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_i, \mathbf{x}))$$

$$\psi_{i-1,i}(y_{i-1}, y_i, \mathbf{x}) = \exp(\sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_{i-1}, y_i, \mathbf{x}))$$

où K_1 et K_2 sont respectivement le nombre de fonctions $f_k^{(1)}$ et $f_k^{(2)}$.

Ces fonctions de caractéristiques $f_k^{(1)}$ et $f_k^{(2)}$ sont donc définies sur les cliques du graphe d'indépendances. Dans le cas présent des chaînes du premier ordre, ces cliques sont toutes les paires de deux nœuds consécutifs, ainsi que tous les nœuds individuellement.

Les fonctions de caractéristiques définies sur ces cliques sont des fonctions qui prennent en paramètres les labels affectés aux variables aléatoires de la clique ainsi que l'ensemble de l'observation $\mathbf{X}$. Ces fonctions sont des fonctions à valeurs réelles. Elles sont définies comme suit :

$$f^{(1)} : \mathcal{Y} \times \times \mathcal{X}^n \to \mathbb{R}$$

$$f^{(2)} : \mathcal{Y}^2 \times \mathcal{X}^n \to \mathbb{R}$$

Celles-ci ont pour but de guider le champ aléatoire conditionnel dans le choix des labels au niveau des cliques. Bien qu'étant des fonctions à valeurs réelles, ces fonctions sont le

$$f^{(1)}(y_i, \mathbf{x}) = \begin{cases} 1 & \text{si } y_i = \texttt{author} \text{ et } x_i \text{ est souligné} \\ 0 & \text{sinon} \end{cases}$$

$$f^{(2)}(y_{i-1}, y_i, \mathbf{x}) = \begin{cases} 1 & \text{si } y_{i-1} = \texttt{description}, \, y_i = \texttt{description} \\ & \text{et } x_{i-2} = \text{"writes"} \\ 0 & \text{sinon} \end{cases}$$

FIG. 3.16 – Exemples de fonctions de caractéristiques pour une tâche d'annotation syntaxique.

plus souvent binaires, renvoyant 1 quand un comportement est observé et 0 dans le cas contraire. La figure 3.16 présente des exemples de fonctions de caractéristiques pour la tâche de transformation de l'arbre XHTML de la page Web de la figure 3.14.

La première fonction s'applique sur les cliques constituées d'un seul nœud. Elle retourne 1 si le mot à la position i dans l'observation $\mathbf{x}$ est souligné et si ce mot a été annoté par `author`. La seconde fonction de caractéristiques, elle, s'applique sur les cliques constituées de 2 nœuds consécutifs. Elle retourne 1 si les mots aux positions $i-1$ et i ont tous deux été annotés par `description` et si le mot les précédant x_{i-2} est "writes". Comme dans le cas des MEMMs, on voit sur cette seconde fonction de caractéristiques que celles-ci permettent des dépendances longue distance sur l'observation $\mathbf{x}$, c'est-à-dire des dépendances sur n'importe quelle partie de l'observation.

À chaque fonction de caractéristiques $f^{(j)}$ est associé un poids $\lambda^{(j)}$. Ce poids est un réel qui va venir influencer le choix des labels lors des tâches d'annotation, de façon à donner une probabilité plus élevée aux annotations correctes. De manière intuitive, plus le poids associé à la première fonction de caractéristiques de la figure 3.16 est élevé, plus le champ aléatoire conditionnel accordera une probabilité élevée aux pages dans lesquelles les mots soulignés ont été annotés par `author`. En règle générale, des poids positifs sont associés aux fonctions représentant un comportement que l'on veut observer dans l'annotation de sortie, tandis qu'un poids négatif est attribué aux fonctions représentant une "mauvaise" annotation.

Avec ces définitions, la probabilité conditionnelle d'une annotation $\mathbf{y}$ sachant une observation $\mathbf{x}$ dans un champ aléatoire conditionnel s'exprime :

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z} \prod_{i=1}^{n} \psi_i(y_i, \mathbf{x}) \prod_{i=2}^{n} \psi_{i-1,i}(y_{i-1}, y_i, \mathbf{x}) \\ p(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z} \prod_{i=1}^{n} \exp\Big(\sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_i, \mathbf{x}) \Big) \prod_{i=2}^{n} \exp\Big(\sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_{i-1}, y_i, \mathbf{x}) \Big) \\ p(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z} \exp\Big(\sum_{i=1}^{n} \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_i, \mathbf{x}) + \sum_{i=2}^{n} \sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_{i-1}, y_i, \mathbf{x}) \Big) \end{aligned} \tag{3.17}$$

Le coefficient de normalisation Z ne dépend ici que de l'observation $\mathbf{x}$. Il est calculé,

dans le cas présent, en sommant sur les probabilités non normalisées pour toutes les annotations $\mathbf{y}$ possibles :

$$Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}^n} \exp\Big(\sum_{i=1}^{n} \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_i, \mathbf{x}) + \sum_{i=2}^{n} \sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_{i-1}, y_i, \mathbf{x}) \Big) \tag{3.18}$$

3.3.2 Algorithmes pour les champs aléatoires conditionnels

Les champs aléatoires conditionnels étant un modèle pour apprendre à annoter, deux problèmes majeurs se posent. Le premier est celui de la recherche de la meilleure annotation étant donnés en paramètres une observation et un CRF (les fonctions de caractéristiques et leurs poids). Ce problème correspond à la recherche du maximum a posteriori (MAP). Le second problème de **l'estimation des paramètres** consiste à trouver les paramètres qui maximisent la vraisemblance d'un ensemble d'apprentissage composé de couples (observation,annotation). Pour ces deux problèmes, il existe des algorithmes efficaces spécifiques aux séquences. Nous les présentons maintenant.

Algorithme de recherche du maximum a posteriori

Nous abordons donc dans un premier temps le problème de la recherche de l'annotation la plus probable pour une observation donnée, aussi appelé recherche du **maximum a posteriori**. Ce problème prend en paramètres une observation $\mathbf{x}$ ainsi qu'un champ aléatoire conditionnel, c'est-à-dire un ensemble de fonctions de caractéristiques et leurs poids associés, et retourne en sortie l'annotation $\hat{\mathbf{y}}$ qui maximise la probabilité $p(\mathbf{y}|\mathbf{x})$:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}^n} p(\mathbf{y}|\mathbf{x})$$

L'algorithme naïf de recherche du maximum a posteriori étant exponentiel dans la taille de la séquence à annoter, un algorithme de programmation dynamique, adapté de l'algorithme de Viterbi utilisé dans les modèles de Markov cachés (*cf.* section 3.2.2.0), et par conséquent un cas particulier de l'algorithme *max-product* (*cf.* section 3.1.5.0), permet de réaliser cette recherche efficacement. Pour cela, on définit la variable $\delta_i(y_i)$. Celle-ci correspond ici au logarithme de la probabilité non normalisée de la meilleure annotation possible de la sous-chaîne $x_1 \dots x_i$, où x_i est annoté par y_i :

$$\begin{aligned}
\delta_i(y_i) &= \max_{y_1 \dots y_{i-1} \in \mathcal{Y}^{i-1}} \log\Big(\prod_{j=1}^{i} \exp\Big(\sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_j, \mathbf{x}) \Big) \prod_{j=2}^{i} \exp\Big(\sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_{j-1}, y_j, \mathbf{x}) \Big) \Big) \\
&= \max_{y_1 \dots y_{i-1} \in \mathcal{Y}^{i-1}} \sum_{j=1}^{i} \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_j, \mathbf{x}) + \sum_{j=2}^{i} \sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_{j-1}, y_j, \mathbf{x})
\end{aligned}$$

On voit ainsi qu'en travaillant au logarithme, on s'est affranchi du calcul des exponentielles intrinsèques aux CRFs. Cela permet à la fois une définition plus simple de la variable δ, mais garantit aussi une meilleure efficacité lors de l'implantation de ces calculs dans un

programme. Avec cette simplification, la définition récursive de la variable δ est donc la suivante :

$$\begin{aligned}
\text{Initialisation :} \qquad \delta_1(y_1) &= \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_1, 1, \mathbf{x}) \\
\text{Récursion :} \qquad \delta_{i+1}(y_{i+1}) &= \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_{i+1}, i+1, \mathbf{x}) \\
&\quad + \max_{y_i \in \mathcal{Y}} \left(\delta_i(y_i) + \sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_i, y_{i+1}, i+1, \mathbf{x}) \right) \\
\text{Fin :} \qquad p(\hat{\mathbf{y}}|\mathbf{x}) &= \frac{1}{Z(\mathbf{x})} \exp\left(\max_{y_n \in \mathcal{Y}} \delta_n(y_n) \right)
\end{aligned}$$

Le score de la meilleure annotation possible est donc $p(\hat{\mathbf{y}}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\max_{y_n \in \mathcal{Y}} \delta_n(y_n) \right)$ et la meilleure annotation $\hat{\mathbf{y}}$ correspond donc au chemin de Viterbi qui a été mémorisé au fur et à mesure des calculs des variables δ.

Chaque variable δ est mémorisée et n'est calculée qu'une seule fois. À chaque position dans la séquence à annoter, $|\mathcal{Y}|$ variables sont calculées, ce qui fait un total de $n \times |\mathcal{Y}|$ valeurs à calculer. Chacune des variables δ contenant le calcul d'un maximum sur l'ensemble des labels dans l'alphabet $\mathcal{Y}$, la complexité en temps de cette algorithme est donc en $\mathcal{O}(n \times |\mathcal{Y}|^2)$.

Estimation des paramètres

Définition 3.9 *Le problème de l'estimation des paramètres Λ prend en entrée un ensemble $S = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$ de couples (observation,annotation) et un ensemble de fonctions de caractéristiques, et retourne en sortie le vecteur Λ des poids associés à ces fonctions qui maximise un critère d'optimisation.*

Avec les champs aléatoires conditionnels, le critère d'optimisation le plus couramment utilisé est le maximum de vraisemblance. L'objectif est de trouver les paramètres du système, c'est-à-dire les poids des fonctions de caractéristiques, qui maximisent la vraisemblance de l'échantillon d'apprentissage S. La **vraisemblance**, aussi appelée **vraisemblance conditionnelle**, d'un échantillon d'apprentissage S, étant donné un vecteur de paramètres Λ, s'exprime de la façon suivante :

$$L(S, \Lambda) = \prod_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in S} p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) \tag{3.19}$$

où p_Λ est la probabilité conditionnelle définie par le champ aléatoire conditionnel dont le vecteur de paramètres est Λ. Étant un produit de probabilités, la fonction de vraisemblance L est donc définie sur l'intervalle $[0, 1]$.

Cette fonction de vraisemblance est utilisée pour juger de la qualité du modèle par rapport aux données d'apprentissage fournies. Comme dans le cas de la recherche du

maximum a posteriori, afin de simplifier les calculs lors de l'estimation des paramètres, on travaille avec la log-vraisemblance :

$$\begin{aligned}\mathcal{L}(S,\Lambda) &= \log(L(S,\Lambda)) \\ &= \log\Big(\prod_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})\Big) \\ &= \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \log p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})\end{aligned} \tag{3.20}$$

La fonction de log-vraisemblance $\mathcal{L}$ est le logarithme d'une fonction définie sur l'intervalle $[0,1]$. $\mathcal{L}$ est donc une fonction strictement négative, définie sur l'intervalle $]-\infty,0]$. De plus, dans le cas des champs aléatoires conditionnels, on a la propriété suivante :

Propriété 3.2 *Dans les champs aléatoires conditionnels, étant donné un échantillon d'apprentissage* $S = \{(\mathbf{x}^{(1)},\mathbf{y}^{(1)}),\ldots,(\mathbf{x}^{(m)},\mathbf{y}^{(m)})\}$*, la fonction de log-vraisemblance* $\mathcal{L}(S,\Lambda)$ *est concave.*

Cette propriété garantit l'existence d'une unique solution $\hat{\Lambda}$ optimale au problème de maximisation de la log-vraisemblance.

Le principe du maximum de vraisemblance consiste donc à trouver les paramètres Λ faisant en sorte que $\mathcal{L}$ soit le plus proche de 0 possible, afin d'avoir une distribution de probabilité p_Λ ayant le plus probablement possible généré les exemples de l'échantillon d'apprentissage S. Toutefois, avec un tel critère d'apprentissage, les risques de surapprentissage sont importants. En effet, la distribution de probabilité apprise peut être trop proche de l'échantillon d'apprentissage, ne permettant alors pas d'annoter d'autres données. Pour cela, la log-vraisemblance est souvent pénalisée (ou régularisée). Le principe de cette régularisation est de pénaliser les vecteurs de paramètres Λ dont la norme $\|\Lambda\|$ est trop élevée. Un choix de pénalisation classique est :

$$\mathcal{L}(S,\Lambda) = \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \log p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2} \tag{3.21}$$

Le paramètre σ détermine combien les paramètres élevés doivent être pénalisés. Le choix de ce paramètre peut être difficile. Bien que cette pénalisation soit couramment utilisée, afin de simplifier les notations, nous considérons dans la suite la log-vraisemblance non pénalisée. Cela ne change en rien les algorithmes.

Lorsqu'on développe la formule de log-vraisemblance en utilisant la formule de probabilité conditionnelle dans un champ aléatoire conditionnel, on obtient l'équation (3.22). Dans un souci de lisibilité des formules, on considère dans la suite de cette section que les fonctions de caractéristiques sur les cliques à 1 nœud $f_k^{(1)}(y_i,\mathbf{x})$ sont un cas particulier des fonctions de caractéristiques sur les cliques à deux nœuds. De plus, on note :

$$\lambda^{(2)}.\mathbf{f}_i^{(2)} = \sum_{k=1}^{K_2} \lambda_k^{(2)} f(y_{i-1}, y_i, i, \mathbf{x})$$

$$\begin{aligned}\mathcal{L}(S,\Lambda) &= \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \log p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) \\ &= \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \Big(\sum_{i=2}^{n} \lambda^{(2)}.\mathbf{f}_i^{(2)} - \log Z(\mathbf{x})\Big) \\ &= \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \Big(\sum_{i=2}^{n} \lambda^{(2)}.\mathbf{f}_i^{(2)} - \log\big(\sum_{\mathbf{y}\in\mathcal{Y}^n} \exp(\sum_{i=2}^{n} \lambda^{(2)}.\mathbf{f}_i^{(2)})\big)\Big)\end{aligned} \tag{3.22}$$

L'estimation des paramètres nécessite donc la maximisation de la fonction de log-vraisemblance $\mathcal{L}(S,\Lambda)$. Toutefois, puisqu'il est impossible de dériver cette fonction par rapport à l'ensemble des paramètres Λ, il n'existe pas de solution analytique. Pour maximiser la log-vraisemblance, diverses techniques d'optimisation approchée sont donc à notre disposition, parmi lesquelles les méthodes à base de gradient sont les plus utilisées. On compte parmi celles-ci les classiques méthodes de Newton, dont le gradient conjugué et les L-BFGS [Byrd et al., 1995]. Dans les travaux de [Wallach, 2002], ces méthodes ont été comparées expérimentalement dans le cas des champs aléatoires conditionnels sur les séquences. Cette comparaison a montré un large avantage en nombre d'itérations requis et en temps de calcul pour les L-BFGS.

Ces méthodes à base de gradient nécessitent le calcul de la dérivée partielle de la log-vraisemblance par rapport à chaque paramètre $\lambda_k^{(2)}$ du champ aléatoire conditionnel. Elle est calculée comme suit.

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\lambda_k^{(2)}} &= \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \sum_{i=1}^{n} f_k^{(2)}(y_{i-1},y_i,\mathbf{x}) - \frac{\sum_{\mathbf{y}\in\mathcal{Y}^n}\sum_{i=1}^{n} f_k^{(2)}(y_{i-1},y_i,\mathbf{x})\exp(\sum_{i=2}^{n}\lambda^{(2)}.\mathbf{f}_i^{(2)})}{Z(\mathbf{x})} \\ &= \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \sum_{i=1}^{n} f_k^{(2)}(y_{i-1},y_i,\mathbf{x}) - \sum_{\mathbf{y}\in\mathcal{Y}^n}\sum_{i=1}^{n} f_k^{(2)}(y_{i-1},y_i,\mathbf{x})p(\mathbf{y}|\mathbf{x})\end{aligned} \tag{3.23}$$

La complexité du calcul de cette dérivée partielle est exponentielle dans la longueur de la séquence à annoter, en raison de la somme sur toutes les annotations possibles parmi $\mathcal{Y}^n$. Il est toutefois possible de la réécrire de la façon suivante en utilisant les probabilités marginales :

$$\frac{\partial\mathcal{L}}{\partial\lambda_k^{(2)}} = \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \sum_{i=1}^{n} f_k^{(2)}(y_{i-1},y_i,\mathbf{x}) - \sum_{i=1}^{n}\sum_{(y_{i-1},y_i)\in\mathcal{Y}^2} f_k^{(2)}(y_{i-1},y_i,\mathbf{x})p(y_{i-1},y_i|\mathbf{x}) \tag{3.24}$$

Dans cette formule, la difficulté réside alors dans le calcul de la probabilité marginale $p(y_{i-1},y_i|\mathbf{x})$. Or, les modèles graphiques fournissent un algorithme d'inférence exacte efficace pour le calcul des probabilités marginales, ainsi que pour le calcul de $Z(\mathbf{x})$, appelé algorithme *sum-product* (*cf.* section 3.1.5.0). Nous décrivons maintenant cet algorithme dans

le cas précis des champs aléatoires conditionnels sur les chaînes du premier ordre. Celui-ci est une adaptation de l'algorithme *forward-backward* [Manning and Schütze, 1999] utilisé dans les modèles de Markov cachés.

Le principe de cet algorithme est le même que celui de l'algorithme de Viterbi, à la différence que la fonction max est ici remplacée par une fonction de somme. On définit ainsi la variable $\alpha_i(y_i)$, appelée variable *forward*, comme suit :

$$\alpha_i(y_i) = \sum_{y_1 \dots y_{i-1} \in \mathcal{Y}^{i-1}} \prod_{j=2}^{i} \exp\left(\lambda^{(2)}.\mathbf{f}_j^{(2)}\right)$$

c'est-à-dire, $\alpha_i(y_i)$ est la somme des probabilités non normalisées de toutes les annotations possibles de la sous-séquence $x_1 \dots x_i$, où x_i est annotée par y_i. De la même façon que pour le calcul de la variable δ de l'algorithme *max-product*, la définition récursive de la variable α est la suivante :

$$\begin{aligned}
\text{Initialisation} &: \alpha_1(y_1) = 1 \\
\text{Récursion} &: \alpha_{i+1}(y_{i+1}) = \sum_{y_i \in \mathcal{Y}} \left(\alpha_i(y_i) \exp(\lambda^{(2)}.\mathbf{f}_{i+1}^{(2)})\right) \\
\text{Fin} &: Z(\mathbf{x}) = \sum_{y_n \in \mathcal{Y}} \alpha_n(y_n)
\end{aligned}$$

Il est intéressant de noter que les variables *forward* fournissent un moyen efficace de calculer le coefficient de normalisation $Z(\mathbf{x})$.

Afin de calculer la probabilité marginale $p(y_{i-1}, y_i|\mathbf{x})$, on définit de la même manière les variables *backward*. Celles-ci sont définies comme suit :

$$\beta_i(y_i) = \sum_{y_{i+1} \dots y_n \in \mathcal{Y}^{n-i}} \prod_{j=i+1}^{n} \exp\left(\lambda^{(2)}.\mathbf{f}_j^{(2)}\right)$$

c'est-à-dire, $\beta_i(y_i)$ est la somme des probabilités non normalisées de toutes les annotations possibles de la sous-séquence $x_{i+1} \dots x_n$, sachant que x_i est annoté par y_i. Une fois de plus, il existe une définition récursive des variables *backward* :

$$\begin{aligned}
\text{Initialisation} &: \beta_n(y_n) = 1 \\
\text{Récursion} &: \beta_i(y_i) = \sum_{y_{i+1} \in \mathcal{Y}} \left(\beta_{i+1}(y_{i+1}) \exp(\lambda^{(2)}.\mathbf{f}_{i+1}^{(2)})\right) \\
\text{Fin} &: Z(\mathbf{x}) = \sum_{y_1 \in \mathcal{Y}} \beta_1(y_1)
\end{aligned}$$

On notera que, tout comme les variables *forward*, les variables *backward* permettent elles aussi le calcul du coefficient de normalisation $Z(\mathbf{x})$.

À l'aide des variables *forward* et *backward*, il est donc maintenant possible de calculer

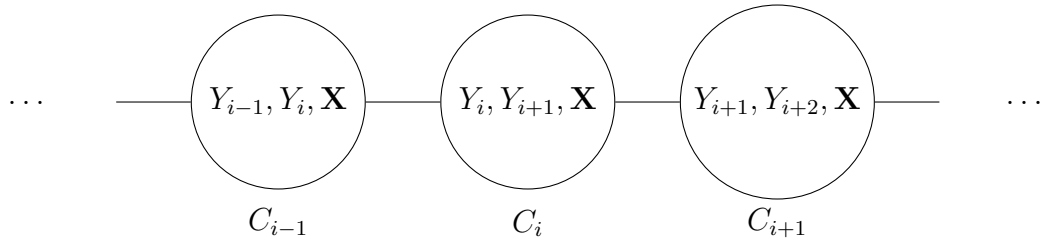

FIG. 3.17 – Arbre de jonction pour les CRFs pour les séquences.

les probabilités marginales. En effet, par définition, on a :

$$\begin{aligned}\alpha_i(y_i)\beta_i(y_i) &= \sum_{y_1 \dots y_{i-1} \in \mathcal{Y}^{i-1}} \prod_{j=2}^{i} \exp\left(\lambda^{(2)}.\mathbf{f}_j^{(2)}\right) \times \sum_{y_{i+1} \dots y_n \in \mathcal{Y}^{n-i}} \prod_{j=i+1}^{n} \exp\left(\lambda^{(2)}.\mathbf{f}_j^{(2)}\right) \\ &= \sum_{y_1 \dots y_{i-1} y_{i+1} \dots y_n \in \mathcal{Y}^{n-1}} \prod_{j=2}^{n} \exp\left(\lambda^{(2)}.\mathbf{f}_j^{(2)}\right) \\ &= Z(\mathbf{x}) p(y_i|\mathbf{x})\end{aligned}$$

Il est alors possible d'exprimer la probabilité marginale de y_i de la façon suivante :

$$p(y_i|\mathbf{x}) = \frac{\alpha_i(y_i)\beta_i(y_i)}{Z(\mathbf{x})} \tag{3.25}$$

Et, de la même façon, on peut exprimer la probabilité marginale $p(y_{i-1}, y_i|\mathbf{x})$ nécessaire au calcul des dérivées partielles comme suit :

$$p(y_{i-1}, y_i|\mathbf{x}) = \frac{\alpha_{i-1}(y_{i-1}) \exp\left(\lambda^{(2)}.\mathbf{f}_i^{(2)}\right) \beta_i(y_i)}{Z(\mathbf{x})}$$

Avec l'aide des variables *forward* et *backward*, la complexité du calcul d'une dérivée partielle est donc de l'ordre de $\mathcal{O}(n \times |\mathcal{Y}|^2)$. La complexité de l'apprentissage, quant à elle, dépend en plus du nombre d'itérations de la descente de gradient utilisée, et est de $\mathcal{O}(n \times K \times |\mathcal{Y}|^2)$.

On peut aussi noter que cet algorithme est une instanciation de l'algorithme *sum-product* sur les modèles graphiques. En effet, les variables *forward* et *backward* correspondent ici aux messages transmis le long de l'arbre de jonction pour les chaînes du premier ordre. En effet, cet arbre de jonction, représenté sur la figure 3.17, est une séquence de nœuds correspondant aux cliques de la forme $\{Y_i, Y_{i+1}, \mathbf{X}\}$, que nous notons ici C_i. Sur cette figure, on a deux types de messages : de gauche à droite et inversement. Dans un premier temps, en utilisant la définition des messages de l'équation (3.7), on a

donc :

$$\begin{aligned}\mu_{C_iC_{i+1}}(y_{i+1}) &= \sum_{y_i} \phi_{C_i}(y_i, y_{i+1})\mu_{C_{i-1}C_i}(y_i) \\ &= \sum_{y_i} \exp\left(\lambda^{(2)}.\mathbf{f}^{(2)}_{i+1}\right)\mu_{C_{i-1}C_i}(y_i) \\ &= \alpha_{i+1}(y_{i+1})\end{aligned}$$

Les variables *forward* correspondent donc aux messages passés de gauche à droite. De la même façon, on a :

$$\begin{aligned}\mu_{C_iC_{i-1}}(y_i) &= \sum_{y_{i+1}} \exp\left(\lambda^{(2)}.\mathbf{f}^{(2)}_{i+1}\right)\mu_{C_{i+1}C_i}(y_{i+1}) \\ &= \beta_i(y_i)\end{aligned}$$

Les variables *backward* correspondent alors aux messages passés de la droite vers la gauche.

3.3.3 Correspondances entre les linear-chain CRFs et les HMMs

Nous étudions maintenant l'expressivité des champs aléatoires conditionnels sur les chaînes du premier ordre. Pour cela, nous allons prouver la proposition suivante :

Proposition 3.1 *Une distribution de probabilité conditionnelle représentée par un modèle de Markov caché peut être représentée par un champ aléatoire conditionnel sur les chaînes du premier ordre.*

Si l'on résume la présentation des modèles de Markov cachés de la section 3.2.2.0, ceux-ci sont définis par :

- des transitions d'un état s à un état t avec leur probabilité $p(t|s)$.
- des émissions d'une observation k à un état s avec leur probabilité $p(k|t)$.
- les probabilités $p(s)$ pour chaque état s d'être un état initial.

Pour représenter la distribution de probabilité définie par un modèle de Markov caché avec un champ aléatoire conditionnel, nous allons représenter ces trois caractéristiques par des fonctions de caractéristiques. Nous illustrerons cette preuve sur l'exemple de HMM de la section 3.2.2.0 que nous rappelons dans la figure 3.18.

Dans un premier temps, pour chaque transition d'un état s à un état t dans le modèle de Markov caché, on crée une fonction de caractéristiques $f_{s,t}$ de la forme :

$$f_{s,t}(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{si } y_{i-1} = s \text{ et } y_i = t \\ 0 & \text{sinon} \end{cases}$$

Les champs aléatoires conditionnels étant un modèle exponentiel, le poids associé à chaque fonction est par conséquent le logarithme de la probabilité de la transition qu'elle représente. On a donc :

$$\forall s, t \in \mathcal{Y}, \lambda_{s,t} = \log p(t|s)$$

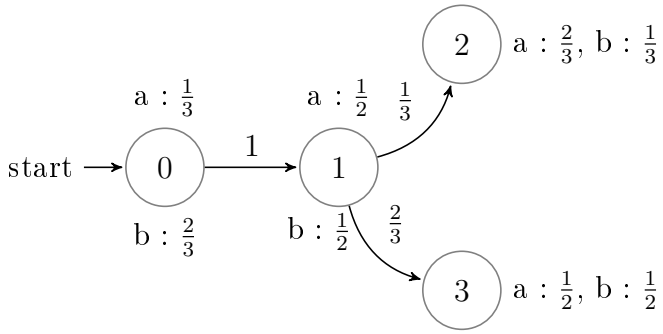

FIG. 3.18 – Exemple de modèle de Markov caché.

Ainsi, sur le HMM de la figure 3.18, la transition de l'état 0 vers l'état 1 est représentée par une fonction de caractéristiques $f_{0,1}$ dont le poids est $\lambda_{0,1} = \log(1) = 0$.

Ensuite, pour chaque émission d'une observation u à un état s dans le modèle de Markov caché, on crée une fonction de caractéristiques $g_{u,s}$ de la forme :

$$g_{u,s}(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{si } x_i = u \text{ et } y_i = s \\ 0 & \text{sinon} \end{cases}$$

De la même façon que précédemment, le poids associé à chaque fonction correspondant à une émission est le logarithme de la probabilité de cette émission :

$$\forall u \in \mathcal{X}, \forall s \in \mathcal{Y}, \mu_{u,s} = \log p(u|s)$$

Sur l'exemple de HMM, l'émission du caractère `a` à l'état 1 est donc représentée par la fonction de caractéristiques $g_{a,1}$ dont le poids est $\mu_{a,1} = \log(\frac{1}{2})$.

Enfin, pour représenter le fait qu'un état s est un état initial, on crée une fonction de caractéristiques de la forme :

$$h_s(y_i, x, i) = \begin{cases} 1 & \text{si } y_i = s \text{ et } i = 1 \\ 0 & \text{sinon} \end{cases}$$

Le poids associé à chacune de ces fonctions est :

$$\forall s \in \mathcal{Y}, \nu_s = \log p(s)$$

Sur notre exemple, on a donc, pour l'état 0, une fonction de caractéristiques h_0 dont le poids est $\nu_0 = \log(1) = 0$.

Nous allons maintenant montrer que la probabilité conditionnelle définie avec un modèle de Markov caché et celle modélisée par un champ aléatoire conditionnel défini comme précédemment sont identiques. Commençons par la probabilité conditionnelle dans un HMM :

$$p_{hmm}(\mathbf{y}|\mathbf{x}) = \frac{p_{hmm}(\mathbf{x}, \mathbf{y})}{p_{hmm}(\mathbf{x})} = \frac{p_{hmm}(\mathbf{x}, \mathbf{y})}{\sum_{\mathbf{y} \in \mathcal{Y}^n} p_{hmm}(\mathbf{x}, \mathbf{y})}$$

Or, en reprenant la formule 3.17, la probabilité dans un champ aléatoire conditionnel sur les séquences s'exprime :

$$p(\mathbf{y}|\mathbf{x}) = \frac{\exp\left(\sum_{i=1}^{n}\sum_{k=1}^{K_1}\lambda_k^{(1)} f_k^{(1)}(y_i, i, \mathbf{x}) + \sum_{i=2}^{n}\sum_{k=1}^{K_2}\lambda_k^{(2)} f_k^{(2)}(y_{i-1}, y_i, i, \mathbf{x})\right)}{\sum_{\mathbf{y} \in \mathcal{Y}^n} \exp\left(\sum_{i=1}^{n}\sum_{k=1}^{K_1}\lambda_k^{(1)} f_k^{(1)}(y_i, i, \mathbf{x}) + \sum_{i=2}^{n}\sum_{k=1}^{K_2}\lambda_k^{(2)} f_k^{(2)}(y_{i-1}, y_i, i, \mathbf{x})\right)}$$

Une condition suffisante, mais pas nécessaire, pour que les distributions de probabilités conditionnelles soient équivalentes est donc que :

$$\exp\Big(\sum_{i=1}^{n}\sum_{k=1}^{K_1}\lambda_k^{(1)}f_k^{(1)}(y_i,i,\mathbf{x})+\sum_{i=2}^{n}\sum_{k=1}^{K_2}\lambda_k^{(2)}f_k^{(2)}(y_{i-1},y_i,i,\mathbf{x})\Big)=p_{hmm}(\mathbf{x},\mathbf{y})$$

Or, à chaque position, seules les fonctions de caractéristiques correspondant aux émissions effectuées sont actives (*ie.* valent 1). À chaque transition, seule la fonction correspondant à la transition courante vaut 1. Enfin, à la première position de la chaîne, la fonction correspondant à l'état initial choisi vaut 1. En remplaçant dans l'équation ci-dessus, on obtient :

$$\begin{aligned}\exp\Big(\nu_{y_1}+\sum_{i=1}^{n}\mu_{x_i,y_i}+\sum_{i=2}^{n}\lambda_{y_{i-1},y_i}\Big)&=\exp\nu_{y_1}.\prod_{i=1}^{n}\exp\mu_{x_i,y_i}.\prod_{i=2}^{n}\exp\lambda_{y_{i-1},y_i}\\&=p(y_1).\prod_{i=1}^{n}p(x_i|y_i).\prod_{i=2}^{n}p(y_i|y_{i-1})\\&=p_{hmm}(\mathbf{x},\mathbf{y})\end{aligned}$$

De plus, dans les HMMs, un certain nombre de transitions et d'émissions de caractères ne sont pas possibles. Par exemple, sur le HMM de la figure 3.18, la transition de l'état 0 à l'état 2 n'est pas permise. Il en est par exemple de même pour l'émission du caractère `c` à l'état 0. Bien qu'elles ne soient pas représentées, ces émissions et transitions ont en fait une probabilité nulle dans le HMM : $p(2|0) = 0$ et $p(c|0) = 0$. Ainsi, elles sont aussi représentées par des fonctions de caractéristiques comme nous les avons décrites précédemment. Le poids qui leur est associé est $\lim_{x\to 0^+}\log(x) = -\infty$.

Les champs aléatoires conditionnels sur les séquences peuvent donc modéliser les mêmes distributions de probabilité conditionnelle que les modèles de Markov cachés.

3.4 Conclusion

Nous avons donc, dans ce chapitre, introduit les modèles graphiques. Ceux-ci permettent de représenter efficacement des distributions de probabilité grâce à l'introduction d'indépendances conditionnelles entre variables aléatoires. De plus, ils sont munis d'algorithmes d'inférence exacte permettant de calculer efficacement les probabilités marginales, ainsi que de trouver le maximum a priori. Parmi ces modèles graphiques, nous en avons présenté trois permettant d'apprendre à effectuer des tâches d'annotations sur les séquences. Parmi ces trois modèles, les champs aléatoires conditionnels semblent les mieux adaptés. En effet, comme nous l'avons prouvé, ils sont au moins aussi expressifs que les modèles de Markov cachés. De plus, ils résolvent le problème du biais de label dont souffrent les MEMMs.

Ce modèle de champs aléatoires conditionnels n'est toutefois pas défini pour un graphe donné et a été étudié dans le cas général, pour des graphes de structure quelconque [Sutton and McCallum, 2006]. C'est pourquoi, dans le cadre de notre problématique de la transformation d'arbres XML par l'annotation, nous nous tournons vers les champs

aléatoires conditionnels. Dans le chapitre suivant, nous adaptons donc les CRFs au cas de l'annotation d'arbres XML.

Chapitre 4

Annotation d'arbres XML avec des champs aléatoires conditionnels

Nous introduisons dans ce chapitre notre adaptation des champs aléatoires conditionnels au problème de l'annotation d'arbres XML.

Nous allons commencer par étudier différents graphes d'indépendances possibles afin de déterminer, selon les tâches abordées et les données utilisées, quels modèles sont les plus adaptés au cas de l'annotation d'arbres XML. Nous mettrons ensuite en relation un de ces modèles avec les automates d'arbres stochastiques binaires. Puis, nous présenterons les algorithmes d'inférence exacte et d'apprentissage pour les champs aléatoires conditionnels adaptés aux arbres XML avant de conclure avec une série d'expériences montrant empiriquement que les champs aléatoires conditionnels sont bien adaptés aux tâches d'annotation d'arbres XML.

4.1 Trois modèles de dépendances pour les arbres XML

Nous commençons donc par discuter le choix du graphe d'indépendances pour l'application des champs aléatoires conditionnels à des tâches d'annotations d'arbres. Ce choix est naturellement fortement dépendant non seulement des tâches abordées, mais aussi des données que nous souhaitons annoter. Pour illustrer nos besoins, nous reprenons l'exemple de la tâche de transformation de l'arbre XHTML de la figure 4.1 en un arbre RSS. Pour cela, on souhaite annoter l'arbre XHTML comme illustré sur la figure 4.2. Pour annoter un tel arbre XML, on associe à chaque nœud n de l'annotation une variable aléatoire Y_n. Sur l'exemple d'annotation, on peut remarquer que de nombreuses dépendances entre ces variables aléatoires sont présentes. En effet, le nœud annoté par `title` se trouve dans le sous-arbre dont la racine est annotée par `item`. On peut donc voir une possible dépendance entre les variables aléatoires correspondant à ces deux nœuds. De la même façon, le nœud annoté par `insert_pubDate` a pour père un nœud annoté par `delete` et est le frère droit d'un nœud annoté par `author`. Il est ici aussi possible de voir des dépendances entres les trois variables aléatoires représentant ces nœuds. Ainsi, pour garder la possibilité d'exprimer des dépendances entre toutes les parties de l'arbre XML, on pourrait vouloir utiliser un graphe d'indépendances complètement connecté, représentant un champ aléatoire $\mathbf{Y}$

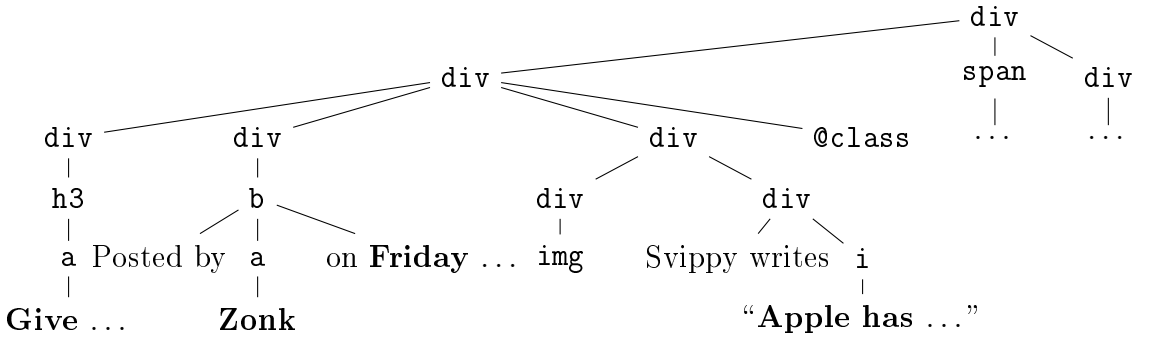

FIG. 4.1 – Exemple d'arbre XHTML pour la tâche de transformation RSS. Cet arbre représente la page Web issue de Slashdot (figure 1.7).

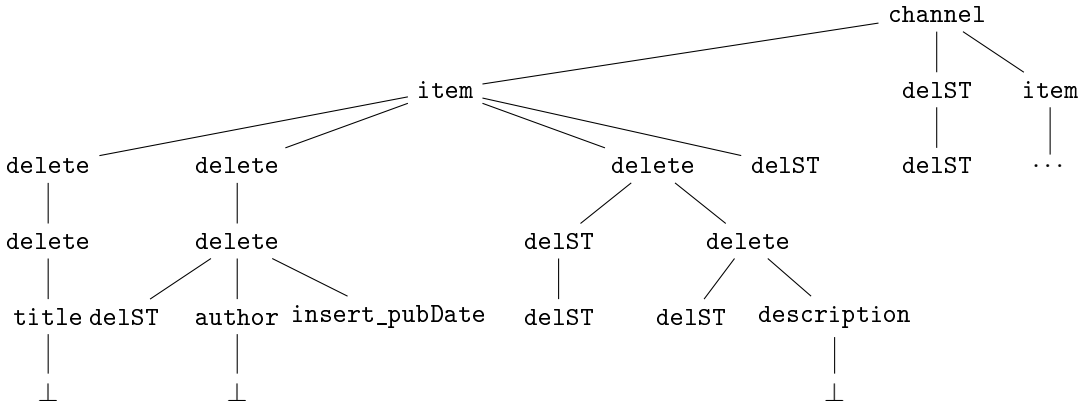

FIG. 4.2 – Annotation "opérations d'édition" de l'arbre XHTML de la figure 4.1 représentant la page Web issue de Slashdot (figure 1.7).

dont toutes les variables aléatoires Y_i seraient interdépendantes. Un tel choix est toutefois impossible car il implique que les algorithmes d'inférence exacte et d'apprentissage que nous voulons mettre en œuvre sur ces tâches d'annotation d'arbres XML deviennent alors exponentiels en la taille des arbres XML.

Il est donc nécessaire de faire des choix. Ceux-ci consistent à définir des indépendances conditionnelles entre variables aléatoires correspondant à l'annotation d'un arbre XML. Ces choix doivent proposer un bon compromis entre une complexité algorithmique suffisamment faible afin de pouvoir appliquer les algorithmes d'inférence et d'apprentissage, et un modèle de dépendances permettant une expressivité suffisante afin de résoudre les tâches d'annotations envisagées. Dans ce but, nous proposons trois choix possibles de graphes d'indépendances, de complexité croissante, en présentant leurs intérêts et défauts.

4.1.1 1-CRFs : Classification indépendante des noeuds

Le premier modèle de dépendances possible, que nous appelons les **1-CRFs** , est le plus simple. Celui-ci consiste à considérer que toutes les variables aléatoires Y_i correspondant aux annotations des nœuds sont indépendantes les unes des autres.

Avec ce modèle, une tâche d'annotation revient à appliquer un même classifieur par maximum d'entropie indépendamment à chaque nœud de l'arbre XML. En effet, les cliques maximales de ce graphe d'indépendances sont constituées d'un seul nœud, et les fonctions de caractéristiques définissables dans ce modèle sont de la forme $f_k^{(1)}(y_n, \mathbf{x}, n)$ où n représente la position du nœud dans l'arbre. Si on note K_1 le nombre de fonctions de caractéristiques utilisées pour le calcul de $\psi_n^{(1)}(y_n, \mathbf{x})$, les fonctions de potentiel sont donc de la forme :

$$\psi_n^{(1)}(y_n, \mathbf{x}) = \exp\Big(\sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_n, \mathbf{x}, n)\Big) \tag{4.1}$$

et la probabilité conditionnelle d'une annotation $\mathbf{y}$ sachant une observation $\mathbf{x}$ s'exprime ainsi :

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{n} \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_n, \mathbf{x}, n)\Big) \tag{4.2}$$

Il est intéressant de noter que le même ensemble de fonctions de caractéristiques et le même vecteur de paramètres $\Lambda^{(1)} = \{\lambda_1, \ldots, \lambda_{K_1}\}$ sont utilisés pour annoter chaque nœud de l'arbre, quelle que soit sa position. Ce choix revient à faire l'hypothèse que tous les nœuds sont annotés selon les même critères.

Ce modèle des 1-CRFs possède l'intérêt d'avoir des algorithmes d'inférence exacte et d'apprentissage de faible complexité. En effet, les cliques maximales du graphe d'indépendances étant de taille 1, ces algorithmes sont à la fois linéaires en la taille des arbres à annoter, mais aussi en la taille de l'alphabet des labels $\mathcal{Y}$.

Toutefois, l'absence totale de dépendances entre les variables aléatoires limite l'expressivité de ce modèle. Il est en effet impossible de modéliser directement des "structures de labels". Sur l'exemple d'annotation de la figure 4.2, on voudrait pouvoir modéliser le fait que le fils d'un nœud annoté par `title` sera très probablement annoté par $\perp$, celui-ci étant le nœud texte contenant le titre. Toutefois, on ne peut modéliser cette dépendance directement à l'aide de fonctions de caractéristiques, puisque celles-ci ne prennent en paramètres qu'un label et l'observation $\mathbf{x}$. Ainsi, la seule information influant sur le choix des labels est ici l'intégralité de l'observation $\mathbf{x}$ elle-même.

4.1.2 2-CRFs : Relation Père-Fils

Le modèle des 1-CRFs, dans sa simplicité, ne tient pas compte de la notion inhérente à tout arbre : la notion de structure hiérarchique. En effet, si l'on revient au cas des champs aléatoires conditionnels sur les séquences, il existe, dans les séquences, une relation de successeur : $\texttt{succ}(i-1) = i$. Cette relation a naturellement été choisie pour modéliser les dépendances entre variables aléatoires dans ce modèle. De la même façon, dans les arbres, une relation de successeur possible est la relation `child`. En effet, dans un arbre, tout nœud possède un père ainsi qu'un ensemble de fils. De plus, dans les arbres XML, cette relation `child`, grâce entre autres à la définition de schéma, possède un sens. Ainsi, par exemple, dans l'arbre de la figure 4.2, on peut supposer qu'il existe un lien fort entre le choix du label `author` pour un nœud et le choix du label $\perp$ pour son fils. Il en est de même pour le choix des labels `delST` qui signifient "suppression du sous-arbre". En effet, si on

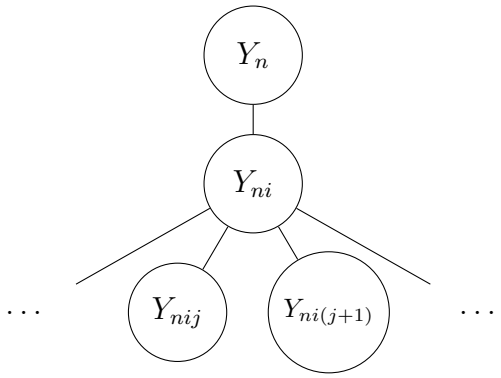

FIG. 4.3 – Graphe d'indépendances des 2-CRFs. Y_{ni} dépend uniquement de son père Y_n et de ses fils.

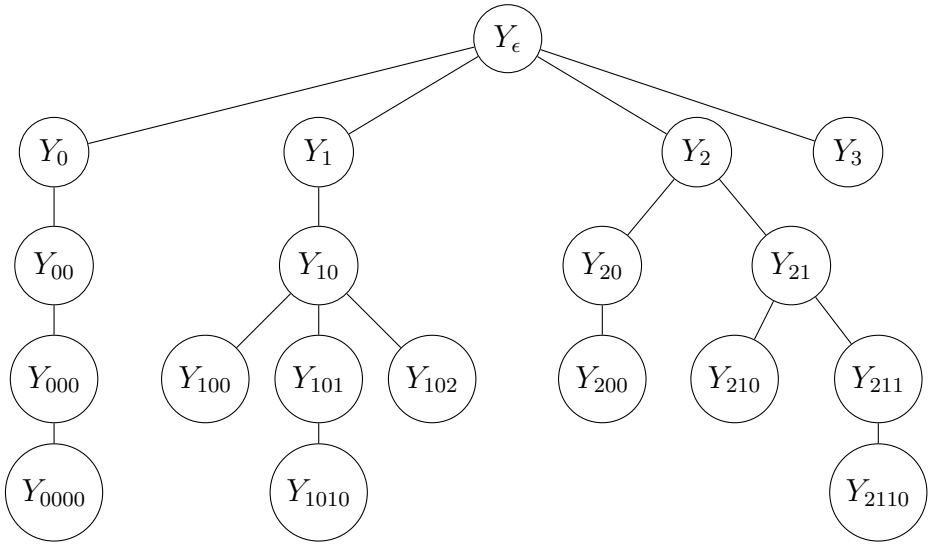

FIG. 4.4 – Graphe d'indépendances du sous-arbre enraciné à `item` de la figure 4.2 avec les 2-CRFs.

supprime le sous-arbre enraciné au nœud n, on supprime aussi les sous-arbres enracinés aux fils de n. Il semble donc intéressant de modéliser cette relation de dépendance. C'est pourquoi, le second modèle de dépendances que nous proposons, appelé **2-CRFs** , ajoute au modèle précédent une relation de dépendance père-fils entre les variables aléatoires de l'annotation. Ce nouveau modèle de dépendance est représenté sur la figure 4.3. Sur celui-ci, on remarque que la variable aléatoire Y_{ni} ne dépend donc que de la variable aléatoire Y_n, le nœud n étant le père de ni, ainsi que des variables aléatoires correspondant aux fils de ni. Avec ce nouveau modèle de dépendances, le graphe d'indépendances pour l'arbre annotation de la figure 4.2 est celui représenté sur la figure 4.4.

Dans le modèle de dépendances des 2-CRFs, on trouve deux types de cliques. On a tout d'abord les mêmes cliques composées d'un unique nœud que dans les 1-CRFs, mais aussi des cliques composées de 2 nœuds : un nœud et son père. Sur ces cliques sont définies des fonctions de caractéristiques de la forme $f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, ni)$, où n correspond à un nœud

dans le graphe et ni au i^{eme} fils de ce nœud. On notera que les cliques "père-fils" sont identifiables de façon unique par la position du fils ni, un nœud d'un arbre ne possédant qu'un unique père. Avec de telles fonctions de caractéristiques, les fonctions de potentiel sur les cliques "père-fils" sont de la forme :

$$\psi_{ni}^{(2)}(y_n, y_{ni}, \mathbf{x}) = \exp\Big(\sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, ni)\Big) \tag{4.3}$$

où K_2 est le nombre de fonctions de caractéristiques $f_k^{(2)}$. La probabilité conditionnelle s'exprime alors :

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{n}\sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_n, \mathbf{x}, n) + \sum_{ni}\sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, ni)\Big) \tag{4.4}$$

Une fois de plus, on notera que pour toutes les cliques de même type (1 ou 2 nœuds), on utilise le même ensemble de fonctions de caractéristiques et les mêmes poids. Cette hypothèse nous permet d'utiliser le même modèle pour annoter des arbres de structures différentes. Ce modèle de champs aléatoires conditionnels pour les arbres est similaire à celui décrit dans [Cohn and Blunsom, 2005].

Du point de vue de la complexité, l'introduction des dépendances "père-fils" a fait passer la taille des cliques maximales à 2. Les algorithmes d'inférence exacte et d'apprentissage, s'ils restent linéaires dans la taille des arbres à annoter, deviennent donc quadratiques dans la taille de l'alphabet de labels. Cette complexité, bien que n'étant pas prohibitive, limite tout de même l'utilisation de ce modèle à des tâches d'annotation avec un nombre raisonnable de labels, que nous estimons empiriquement à 500.

Le principal avantage de ce modèle est que, contrairement aux 1-CRFs, le choix d'un label à toute position n ne dépend plus uniquement de l'observation, mais aussi des labels affectés au père et aux fils de n. Ainsi, sur l'exemple d'annotation de la figure 4.2, on remarque que sous le fils du nœud annoté par `author` est annoté par $\perp$. Ce comportement peut être modélisé par une fonction de caractéristiques de la forme :

$$f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, ni) = \begin{cases} 1 & \text{si } y_n = \texttt{author} \text{ et } y_{ni} = \perp \\ 0 & \text{sinon} \end{cases}$$

En affectant un poids suffisamment élevé à cette fonction, le comportement cité précédemment sera présent lors de l'annotation. De plus, cette fonction de caractéristiques peut aussi comporter un ou plusieurs tests portant sur l'observation $\mathbf{x}$. On pourrait par exemple écrire une fonction plus précise comme :

$$f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, ni) = \begin{cases} 1 & \text{si } y_n = \texttt{author},\ y_{ni} = \perp \\ & \quad \text{et } x_n = \texttt{a} \\ 0 & \text{sinon} \end{cases}$$

4.1.3 3-CRFs : Relation Père-Fils-Frère

Les 2-CRFs ne prennent toutefois pas en compte certaines dépendances naturelles dans les arbres XML. En effet, ces arbres sont partiellement ordonnés. En plus de la relation

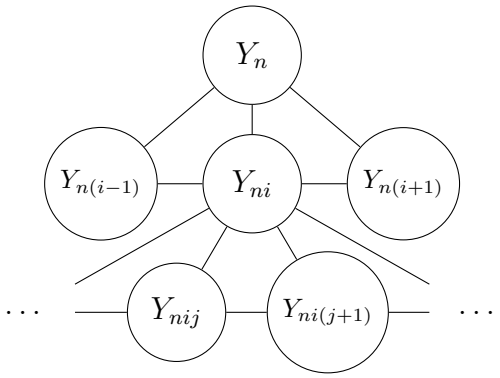

FIG. 4.5 – Graphe d'indépendances des 3-CRFs. Y_{ni} dépend de son père Y_n, de ses frères précédent et suivant, et de ses fils.

`child` entre un nœud et ses fils, il existe aussi la relation de frère suivant `next-sibling` définie dans la section 1.1.1. Ici encore, cette relation est non seulement structurelle, mais a aussi, dans le cas des arbres XML, un intérêt sémantique. En effet, si l'on prend l'exemple de l'annotation de la figure 4.2, on peut supposer qu'il existe une dépendance forte entre les variables aléatoires correspondant aux annotations `author` et `insert_pubDate`. En effet, dans les documents XML à annoter, la date apparaît systématiquement après le nom de l'auteur. L'utilisation de cette dépendance doit donc permettre de mieux résoudre des tâches d'annotation d'arbres XML. Afin de tenir compte de cette relation, nous introduisons donc un troisième modèle de dépendances appelé 3-CRFs qui ajoute au modèle des 2-CRFs une relation de dépendance entre 2 variables aléatoires si elles correspondent des fils consécutifs d'un même nœud. Le graphe d'indépendances des 3-CRFs est représenté sur la figure 4.5. Dans ce modèle, une variable aléatoire Y_{ni} dépend donc de son père Y_n, de ses enfants et de ses frères gauche $Y_{n(i-1)}$ et droit $Y_{n(i+1)}$. Le modèle des 3-CRFs prend toutefois en compte une autre spécificité des arbres XML. En effet, ceux-ci sont des arbres partiellement ordonnés. Ainsi, si les nœuds de type élément sont ordonnés, il n'y a en revanche pas d'ordre sur les attributs. Définir une dépendance de type `next-sibling` entre deux attributs consécutifs ou entre un élément et un attribut n'aurait donc aucun sens. C'est pourquoi les attributs ne dépendent que de leur père et pas des nœuds, élément, attribut ou texte, voisins. En tenant compte de ces spécificités, on obtient ainsi, pour l'arbre annotation de la figure 4.2, le graphe d'indépendances représenté sur la figure 4.6.

On a donc, dans ce modèle, en plus de deux types de cliques déjà présents dans les 2-CRFs, des cliques composées d'un nœud et de son frère suivant, ainsi que des cliques composées d'un nœud, de son père et de son frère suivant, que nous appelons **cliques triangulaires**. On peut donc définir un nouveau type de fonctions de caractéristiques de la forme $f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni)$. Par souci de simplification, nous ne définissons pas de fonction de caractéristiques sur les cliques composées de deux nœuds consécutifs, étant donné que celles-ci sont un cas particulier des fonctions définies sur les cliques triangulaires. En effet, il existe exactement une clique de ce type pour chaque clique triangulaire.

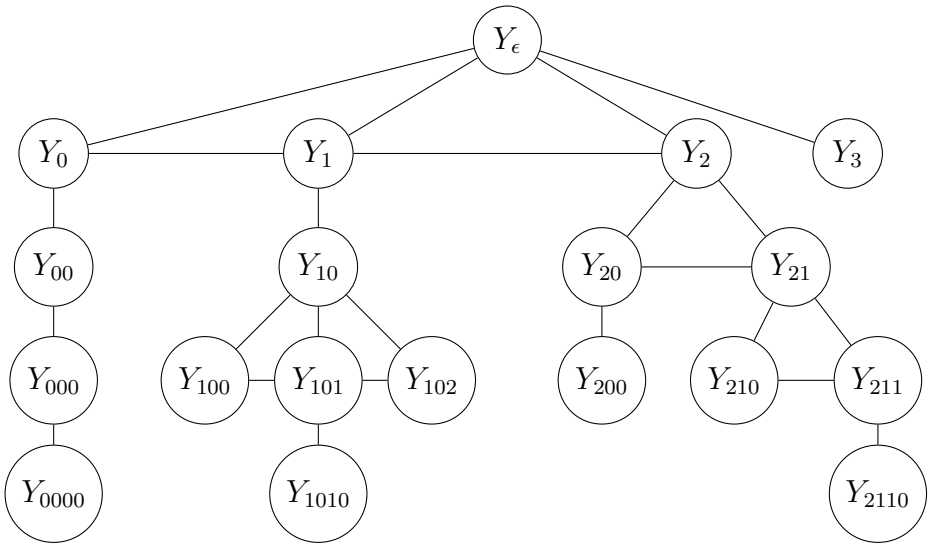

FIG. 4.6 – Graphe d'indépendances du sous-arbre enraciné à `item` de la figure 4.2 avec les 3-CRFs.

Les nouvelles fonctions de potentiel sur ces cliques sont donc de la forme :

$$\psi_{ni}^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}) = \exp\Big(\sum_{k=1}^{K_3} \lambda_k^{(3)} f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni)\Big) \tag{4.5}$$

où K_3 est le nombre de fonctions de caractéristiques $f_k^{(3)}$. La formule de la probabilité conditionnelle dans les 3-CRFs devient donc :

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(&\sum_n \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_n, \mathbf{x}, n) \\ &+ \sum_{ni} \sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, ni) + \sum_{ni} \sum_{k=1}^{K_3} \lambda_k^{(3)} f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni)\Big) \end{aligned} \tag{4.6}$$

Pour simplifier les notations, on suppose que si le nœud $n(i+1)$ n'existe pas, on a

$$\lambda_k^{(3)} f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni) = 0$$

Comme c'était le cas pour les cliques à 1 ou 2 nœuds dans le cas des 2-CRFs, on utilise ici le même ensemble de fonctions de caractéristiques, avec les mêmes poids, quelle que soit la clique triangulaire considérée. Ce modèle de clique triangulaire s'applique ainsi comme un patron pour tout triplet contenant un nœud, son père et son frère suivant.

Les cliques maximales de ce modèle étant de taille 3, la complexité des algorithmes associés est donc maintenant cubique. Plus encore que dans le cas des 2-CRFs, ceci peut être prohibitif pour l'application à des tâches d'annotations avec un très grand nombre de labels. Toutefois, lorsque l'alphabet est de petite taille (moins de 60 labels), ce qui est souvent le cas, ce modèle permet d'exprimer des dépendances bien plus complexes que les

	1-CRFs	2-CRFs	3-CRFs
Taille des cliques maximales	1	2	3
Dépendances	Uniquement l'observation $\mathbf{x}$	Observation + Père-Fils	Observation + Père-Fils-Frère
Complexité en taille	$\mathcal{O}(N \times \lvert\mathcal{Y}\rvert)$	$\mathcal{O}(N \times \lvert\mathcal{Y}\rvert^2)$	$\mathcal{O}(N \times \lvert\mathcal{Y}\rvert^3)$
Nombre maxi de labels	≈ 20000	≈ 500	≈ 60

TAB. 4.1 – Comparaison des 3 modèles de dépendances

deux modèles précédents grâce à l'expressivité des fonctions de caractéristiques définies sur les cliques triangulaires. Nous donnons, dans la section 4.1.4, un exemple de cette expressivité.

Toutefois, l'augmentation de la complexité des algorithmes est contrebalancée par les possibilités d'expression qu'offre le modèle de dépendances des 3-CRFs. En effet, si l'on se réfère à l'exemple de la figure 4.2, on observe qu'il existe un lien fort entre le fait qu'un nœud est annoté par `author` et que son frère suivant est annoté par `insert_pubDate`. De plus leur père est annoté par `delete`. La dépendance forte entre ces trois labels peut être représentée par une fonction de caractéristiques de la forme :

$$f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni) = \begin{cases} 1 & \text{si } y_n = \texttt{delete},\ y_{ni} = \texttt{author}, \\ & y_{n(i+1)} = \texttt{insert\_pubDate} \text{ et } x_{n(i-1)} = \text{“Posted By”} \\ 0 & \text{sinon} \end{cases}$$

Par le biais de fonctions de caractéristiques de ce type, les 3-CRFs peuvent donc exprimer des comportements, c'est-à-dire des affectations de labels aux cliques, que les deux modèles précédents ne pouvaient exprimer. Ce modèle des 3-CRFs, en exploitant plusieurs propriétés des arbres XML, semble donc être le mieux adapté des trois pour résoudre des tâches d'annotation sur ce type d'arbres.

Le tableau 4.1 fournit un résumé comparatif des trois modèles de dépendances que nous venons de définir pour les champs aléatoires conditionnels sur les arbres XML. Celui-ci présente, pour chaque modèle, la taille des cliques maximales, les dépendances modélisées, la complexité en taille mémoire et la taille maximum de l'alphabet des labels que l'on peut utiliser (évaluée empiriquement), compte tenu de cette complexité.

4.1.4 3-CRFs et automates d'arbres binaires stochastiques

Afin de montrer l'intérêt des 3-CRFs du point de vue de l'expressivité du modèle, nous allons établir un parallèle entre les distributions de probabilité définies par ce modèle et celui des automates d'arbres binaires stochastiques, de la même façon que nous l'avions fait pour les champs aléatoires conditionnels sur les séquences et les modèles de Markov cachés (*cf.* section 3.3.3).

Si on ignore les attributs, les arbres XML sont des arbres ordonnés d'arité non bornée. De tels arbres peuvent être représentés sous la forme d'arbres binaires en utilisant des

codages tels que le codage "first-child next-sibling" [Neven, 2002] ou le codage currifié [Carme et al., 2004b]. Les distributions de probabilité sur les arbres d'arité non bornée peuvent donc être définies sur le codage binaire de ces arbres. De telles distributions de probabilités sur les arbres binaires peuvent être définies à l'aide d'automates d'arbres stochastiques descendants. Ceux-ci peuvent aussi être vus comme des grammaires d'arbres régulières stochastiques. Ils définissent une probabilité jointe $p(\mathbf{x}, \mathbf{y})$, où $\mathbf{x}$ est l'observation et $\mathbf{y}$ correspond aux états de l'automate associés à chaque nœud, c'est-à-dire l'annotation. Ces probabilités jointes permettent de définir de façon naturelle une distribution de probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$.

Commençons donc par rappeler la définition d'un automate d'arbres stochastique descendant :

Définition 4.1 *Un* automate d'arbres stochastique descendant *sur un alphabet binaire $\mathcal{X}$ est un triplet $\mathcal{A} = (Q, I, \Delta)$ où Q est un ensemble d'états fini, I est une fonction de Q dans $]0, 1]$ et Δ est un ensemble de règles de transition de la forme :*

$$r : \ q \rightarrow b(q_1, q_2) \tag{4.7}$$

où b est un symbole binaire dans $\mathcal{X}$, $q, q_1, q_2 \in Q$, ou de la forme :

$$r : \ q \rightarrow a \tag{4.8}$$

où a est une constante de $\mathcal{X}$, $q \in Q$ et $w \in]0, 1]$. La fonction I assigne à chaque état de Q sa probabilité d'être un état initial et à chaque règle r de Δ est assignée une probabilité que l'on note $w(r)$. On appelle $l(r)$ la partie gauche d'un règle r. Un automate d'arbre stochastique descendant doit vérifier les propriétés suivantes :

$$\sum_{q \in Q} I(q) = 1 \tag{4.9}$$

$$\forall q \in Q : \qquad \sum_{l(r)=q, r \in \Delta} w(r) = 1 \tag{4.10}$$

Un tel automate permet alors de calculer la probabilité jointe $p(\mathbf{x}, \mathbf{y})$ d'un arbre $\mathbf{x}$ défini sur $\mathcal{X}$ et du *run* $\mathbf{y}$. Un *run* correspond aux états de Q utilisés pour générer l'arbre $\mathbf{x}$. Cette probabilité est calculée en faisant le produit des probabilités $w(r_n)$ des règles utilisées à chaque nœud n, ainsi que de la probabilité $I(q_\epsilon)$, où q_ϵ représente l'état associé à la racine :

$$p_{\mathcal{A}}(\mathbf{x}, \mathbf{y}) = I(q_\epsilon) \prod_{n \in \mathtt{nodes}(\mathbf{x})} w(r_n)$$

Il faut toutefois noter que, malgré les propriétés (4.9) et (4.10), $p_{\mathcal{A}}$ n'est pas toujours une distribution de probabilité jointe, la somme des $p_{\mathcal{A}}(\mathbf{x}, \mathbf{y})$ pour tous les couples $(\mathbf{x}, \mathbf{y})$ possibles n'étant pas nécessairement 1 [Wetherell, 1980].

Cette probabilité jointe est souvent utilisée pour mesurer la probabilité $p_{\mathcal{A}}(\mathbf{x})$ d'un arbre $\mathbf{x}$. Celle-ci est obtenue en sommant les probabilités jointes sur tous les *runs* $\mathbf{y}$ possibles pour $\mathbf{x}$: $p_{\mathcal{A}}(\mathbf{x}) = \sum_{\mathbf{y}} p_{\mathcal{A}}(\mathbf{x}, \mathbf{y})$. Toutefois, un automate d'arbres descendant permet aussi d'effecteur des tâches d'annotations. En effet, pour cela, on considère que

l'ensemble fini d'états Q est l'alphabet des labels. Ainsi, un run $\mathbf{y}$ correspond à une annotation possible. On peut alors définir la probabilité conditionnelle d'une annotation $\mathbf{y}$ sachant l'observation $\mathbf{x}$ de la façon suivante :

$$p_{\mathcal{A}}(\mathbf{y}|\mathbf{x}) = \frac{p_{\mathcal{A}}(\mathbf{x},\mathbf{y})}{p_{\mathcal{A}}(\mathbf{x})} = \frac{p_{\mathcal{A}}(\mathbf{x},\mathbf{y})}{\sum_{\mathbf{y}} p_{\mathcal{A}}(\mathbf{x},\mathbf{y})}$$

On notera que, contrairement au cas de la probabilité jointe $p_{\mathcal{A}}(\mathbf{x},\mathbf{y})$, le mode de calcul de la probabilité conditionnelle $p_{\mathcal{A}}(\mathbf{y}|\mathbf{x})$ garantit bien que, quel que soit $\mathbf{x}$, on a :

$$\sum_{\mathbf{y}} p_{\mathcal{A}}(\mathbf{y}|\mathbf{x}) = 1$$

Proposition 4.1 *Une distribution de probabilités conditionnelles sur des arbres binaires représentée par un automate d'arbre stochastique descendant peut être représentée par un 3-*CRF.

De la même façon que pour la représentation des modèles de Markov cachés à l'aide d'un champ aléatoire conditionnel sur les chaînes du premier ordre (*cf.* section 3.3.3), nous allons transformer les règles de Δ en fonctions de caractéristiques. Tout d'abord, pour chaque règle de la forme $r : \ q \to b(q_1, q_2)$, on définit une fonction de caractéristiques booléenne :

$$f_r(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni) = \begin{cases} 1 & \text{si } y_n = q, y_{ni} = q_1, y_{n(i+1)} = q_2, x_n = b \\ 0 & \text{sinon} \end{cases}$$

À cette fonction est associé un poids $\lambda_r = \log w(r)$. Il est aussi nécessaire de compléter l'ensemble des fonctions de caractéristiques pour tous les b, q, q_1, q_2 possibles qui ne correspondent pas à des règles de Δ. Ces fonctions se voient attribué un poids de $-\infty$.

De la même façon, pour chaque règle de la forme $r : \ q \to a$, on définit une fonction de caractéristiques booléenne :

$$f_r(y_n, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = q, x_n = a \\ 0 & \text{sinon} \end{cases}$$

On associe à cette fonction un poids $\lambda_r = \log w(r)$. On complète l'ensemble des fonctions de caractéristiques pour tous les q, a ne correspondant pas à des règles de Δ. Le poids qui leur est affecté est $-\infty$.

Enfin, pour chaque état $q \in Q$, on définit une fonction de caractéristiques booléenne de la forme :

$$f_q(y_n, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = q \text{ et } n \text{ est la racine} \\ 0 & \text{sinon} \end{cases}$$

Le poids affecté à chacune de ces fonctions est $\lambda_q = \log I(q)$ si $I(q) \neq 0$ et $\lambda_q = -\infty$ sinon.

Avec ces fonctions de caractéristiques et leurs poids, nous avons défini un 3-CRF que nous appelons CRF($\mathcal{A}$). Nous allons montrer que la probabilité conditionnelle $p_{\text{CRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x})$ définie par le 3-CRF est la même que celle définie par l'automate d'arbres $\mathcal{A}$. Pour cela,

considérons un arbre $\mathbf{x}$ et une annotation $\mathbf{y}$ de $\mathbf{x}$ par $\mathcal{A}$. Selon la définition des champs aléatoires conditionnels, on a :

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_n \sum_{k=1}^{K_1} \lambda_k^{(1)} f_k^{(1)}(y_n, \mathbf{x}, n) + \sum_{ni} \sum_{k=1}^{K_2} \lambda_k^{(2)} f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, ni) + \sum_{ni} \sum_{k=1}^{K_3} \lambda_k^{(3)} f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni) \Big)$$

D'une part, à chaque clique maximale de l'arbre, seule une fonction de caractéristiques prend la valeur 1, cette fonction étant celle correspondant à la règle r utilisée à cette même position. De plus, pour la racine de l'arbre, une seule des fonctions de caractéristiques correspondant aux états initiaux prend la valeur 1. En notant r_n la règle appliquée à la position n, la probabilité conditionnelle définie par notre 3-CRF devient donc :

$$\begin{aligned} p_{\text{CRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x}) &= \frac{1}{Z(\mathbf{x})} \exp(\lambda_{y_\epsilon} f_{y_\epsilon}(y_\epsilon, \mathbf{x}, \epsilon)) \prod_n \exp(\lambda_{r_n} f_{r_n}(y_n, y_{n.0}, y_{n.1}, \mathbf{x}, n.0)) \\ &= \frac{1}{Z(\mathbf{x})} \exp(\log I(y_\epsilon)) \prod_n \exp(\log w(r_n)) \\ &= \frac{1}{Z(\mathbf{x})} p_{\mathcal{A}}(\mathbf{x}, \mathbf{y}) \end{aligned}$$

Considérons maintenant un arbre $\mathbf{y}$ de même structure que $\mathbf{x}$ et défini sur Q mais qui ne correspond pas à une annotation permise par l'automate $\mathcal{A}$. Il existe donc une position p dans cet arbre où aucune règle de l'automate ne s'applique. Par définition du 3-CRF, il existe à cette position une fonction de caractéristiques qui prend la valeur 1 et donc le poids est $-\infty$. Il en résulte que la probabilité d'une telle annotation dans le 3-CRF est nulle. Ainsi, le coefficient de normalisation $Z(\mathbf{x})$ correspond bien à la somme des probabilités jointes sur toutes les annotations possibles de $\mathbf{x}$ dans l'automate $\mathcal{A}$: $Z(\mathbf{x}) = p(\mathbf{x})$.

Nous en déduisons donc que :

$$p_{\text{CRF}(\mathcal{A})}(\mathbf{y}|\mathbf{x}) = \frac{p_{\mathcal{A}}(\mathbf{x}, \mathbf{y})}{p_{\mathcal{A}}(\mathbf{x})} = p_{\mathcal{A}}(\mathbf{y}|\mathbf{x}) \tag{4.11}$$

On notera toutefois que la réciproque n'est pas vraie. En effet, s'il est possible de définir un 3-CRF permettant de calculer les mêmes probabilités conditionnelles qu'un automate d'arbres stochastique descendant, il existe des 3-CRFs tels qu'on ne peut définir d'automate d'arbres stochastique descendant correspondant.

4.1.5 Travaux associés

Si les champs aléatoires conditionnels ont été appliqués dans de nombreux domaines, principalement sur des tâches d'annotation de séquences, on trouve peu de travaux les mettant en œuvre dans le cadre de l'annotation d'arbres. Toutefois, dans le domaine du

traitement automatique de la langue, [Cohn and Blunsom, 2005] ont élaboré un modèle de champs aléatoires conditionnels pour les arbres très similaire à notre modèle des 2-CRFs présenté dans la section 4.1.2. Ils ont appliqué ce modèle à la tâche d'annotation des rôles sémantiques de CONLL 2005[5] [Carreras and Marquez, 2005], qui consiste à annoter les rôles sémantiques dans des arbres d'analyse syntaxique de phrases anglaises extraites du Penn Treebank. Le système présenté par [Cohn and Blunsom, 2005] a obtenu des résultats corrects mais tout de même relativement loin de ceux obtenus par les meilleurs systèmes.

Une autre application des champs aléatoires conditionnels concerne l'annotation sémantique de pages Web au format XHTML avec les catégories d'une ontologie. Le système présenté dans [Tang et al., 2006] utilise lui aussi un graphe d'indépendances ne modélisant que les dépendances entre le label d'un nœud et celui de son père et donc comparable aux 2-CRFs. Le système a été évalué pour l'annotation de rapports annuels de la bourse de Shanghai avec une ontologie et obtient des résultats meilleurs que ceux obtenus, sur cette même tâche, par les SVM et les CRFs sur les séquences.

Enfin, des champs aléatoires conditionnels à structure arborescente ont aussi été utilisés dans le domaine du traitement de l'image. Pour cela, [Awasthi et al., 2007] propose de regrouper des zones de pixels en leur ajoutant un père sous la forme d'une variable cachée. Les variables cachées correspondant à chaque zone de pixels sont alors eux-mêmes regroupés en leur ajoutant une variable cachée comme père. En procédant récursivement, la structure du graphe d'indépendances forme donc un arbre. Une fois encore, ce graphe d'indépendances est le même que celui que nous appelons 2-CRFs. Ce modèle obtient des résultats légèrement supérieurs à ceux obtenus par d'autres modèles de CRF sur des tâches de détection d'objets dans les images.

Ces travaux montrent d'une part que, si les champs aléatoires conditionnels ont été peu utilisés dans les arbres, ils s'adaptent toutefois particulièrement à ces structures et permettent d'offrir de très bons résultats. Ils montrent aussi la nouveauté du modèle des 3-CRFs que nous proposons dans la section 4.1.3, aucun travail de recherche ne semblant, à notre connaissance, avoir étudié spécifiquement ce modèle.

4.2 Algorithmes pour les 3-CRFs

Nous présentons maintenant les algorithmes d'inférence exacte et d'estimation des paramètres pour les 3-CRFs. Les 2-CRFs étant un cas particulier des 3-CRFs, ces algorithmes s'adaptent aussi à ce modèle. Afin de simplifier la lecture des formules, nous utiliserons les abréviations suivantes :

$$\begin{aligned}
\psi_n^{(1)} &= \psi_n^{(1)}(y_n, \mathbf{x}, n) \\
\psi_{ni}^{(2)} &= \psi_{ni}^{(2)}(y_n, y_{ni}, \mathbf{x}, ni) \\
\psi_{ni}^{(3)} &= \psi_{ni}^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, ni)
\end{aligned}$$

De plus, dans toutes les formules qui suivent, on note N la taille de l'arbre $\mathbf{x}$ (et donc de l'arbre annotation $\mathbf{y}$), c'est-à-dire son nombre de nœuds.

[5] `http://www.lsi.upc.es/~srlconll`

4.2.1 Calcul de $Z(\mathbf{x})$

Nous nous intéressons dans un premier temps au calcul du coefficient de normalisation $Z(\mathbf{x})$. En effet, le calcul de celui-ci est nécessaire au calcul de la probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$. Ce coefficient est défini comme une somme sur toutes les annotations possibles de $\mathbf{x}$ des probabilités non-normalisées. Pour rappel, dans le cas des 3-CRFs, il s'exprime comme suit :

$$Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}^N} \left(\prod_n \psi_n^{(1)} \prod_{ni} \psi_{ni}^{(2)} \psi_{ni}^{(3)} \right)$$

Comme nous l'avons déjà noté plusieurs fois, un calcul naïf de ce coefficient de normalisation consistant à effectivement sommer sur toutes les annotations $\mathbf{y}$ possibles nécessiterait un temps exponentiel en la taille N de l'arbre $\mathbf{x}$. De la même façon que dans les champs aléatoires conditionnels sur les séquences, ce coefficient peut être calculé efficacement par un algorithme de programmation dynamique. Comme les graphes d'indépendances des 3-CRFs sont de forme arborescente, nous nous inspirons, pour cet algorithme, de ceux utilisés dans les grammaires hors-contexte probabilistes (PCFG) comme l'algorithme *inside-outside* [Manning and Schütze, 1999].

Pour ce faire, nous définissons donc tout d'abord une variable $\beta_n(y_n)$ que nous appelons variable *inside*. Celle-ci correspond à la somme des probabilités non-normalisées de toutes les annotations possibles du sous-arbre de $\mathbf{x}$ enraciné à la position n. Si on note m le nombre de fils du nœud n, la variable *inside* est définie récursivement de la façon suivante :

$$\text{Initialisation} : \beta_n(y_n) = \psi_n^{(1)} \text{ si } n \text{ est une feuille, } ie.\ m = 0$$

$$\text{Récursion} : \beta_n(y_n) = \psi_n^{(1)} \sum_{y_{n1},\ldots,y_{nm} \in \mathcal{Y}^m} \left(\prod_{i=1}^{m} \beta_{ni}(y_{ni}) \psi_{ni}^{(2)} \prod_{i=1}^{m-1} \psi_{ni}^{(3)} \right)$$

$$\text{Fin} : Z(\mathbf{x}) = \sum_{y_\epsilon \in \mathcal{Y}} \beta_\epsilon(y_\epsilon)$$

Si cette première définition résout le problème de la récursion en profondeur, elle n'est toutefois pas satisfaisante. En effet, nous avons ici réduit la complexité de l'algorithme à un algorithme exponentiel en l'arité maximale de l'arbre $\mathbf{x}$. Or, les arbres XML sont des arbres d'arité non bornée. La présence d'une somme sur toutes les annotations possibles des fils du nœud en position n entraîne donc une explosion combinatoire. Afin de contourner cette complexité, les fils d'un nœud formant une chaîne du premier ordre, nous nous inspirons des algorithmes d'inférence de type *forward-backward* (*cf.* section 3.3.2.0) utilisés dans les champs aléatoires conditionnels sur les séquences. Nous introduisons donc une nouvelle variable $\beta'_{n,k}(y_n, y_{n.k})$ que nous appelons variable *backward*. Celle-ci est définie récursivement comme suit :

$$\text{Initialisation} : \beta'_{n,m}(y_n, y_{nm}) = \beta_{nm}(y_{nm}) \psi_{nm}^{(2)}$$

$$\text{Récursion} : \beta'_{n,k}(y_n, y_{nk}) = \beta_{nk}(y_{nk}) \psi_{nk}^{(2)} \sum_{y_{n(k+1)}} \left(\psi_{nk}^{(3)} \beta'_{n,k+1}(y_n, y_{n(k+1)}) \right)$$

$$\text{Fin} : \beta'_{n,1}(y_n, y_{n1}) = \sum_{y_{n2},\ldots,y_{nm} \in \mathcal{Y}^{(m-1)}} \left(\prod_{i=1}^{m} \beta_{ni}(y_{ni}) \psi_{ni}^{(2)} \prod_{i=1}^{m-1} \psi_{ni}^{(3)} \right)$$

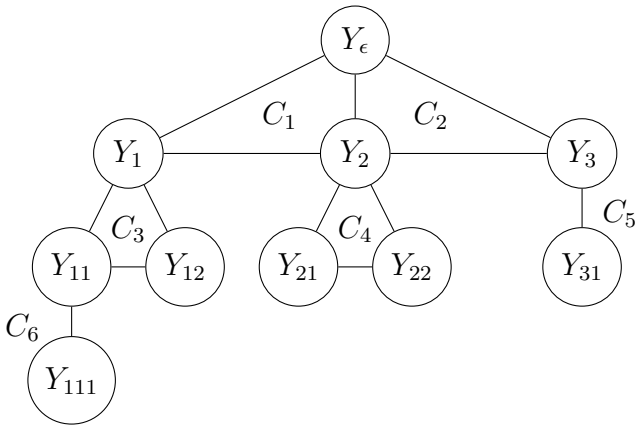

FIG. 4.7 – Exemple de graphe d'indépendances dans le modèle des 3-CRFs.

Avec l'introduction de cette nouvelle variable *backward*, la variable *inside* peut être reformulée de la façon suivante :

$$\begin{aligned}
\text{Initialisation} &: \beta_n(y_n) = \psi_n^{(1)} \text{ si } n \text{ est une feuille, } ie.\ m = 0 \\
\text{Récursion} &: \beta_n(y_n) = \psi_n^{(1)} \sum_{y_{n1} \in \mathcal{Y}} \beta'_{n,1}(y_n, y_{n1}) \\
\text{Fin} &: Z(\mathbf{x}) = \sum_{y_\epsilon \in \mathcal{Y}} \beta_\epsilon(y_\epsilon)
\end{aligned}$$

Le calcul d'une variable *inside*, avec les deux sommes sur tous les labels possibles d'un nœud (une dans le calcul de la variable *inside*, une autre dans le calcul de la variable *backward*), a donc une complexité quadratique dans la taille de l'alphabet des labels $\mathcal{Y}$. D'après la définition de cette variable, on en déduit que le coefficient de normalisation $Z(\mathbf{x})$ peut être réécrit de la façon suivante :

$$Z(\mathbf{x}) = \sum_{y_\epsilon \in \mathcal{Y}} \beta_\epsilon(y_\epsilon) \tag{4.12}$$

La complexité du calcul de $Z(\mathbf{x})$ devient donc cubique dans la taille de l'alphabet des labels $\mathcal{Y}$ et reste linéaire dans la taille de l'arbre $\mathbf{x}$ observé : $\mathcal{O}(N \times |\mathcal{Y}|^3)$.

On peut aussi remarquer que, comme dans le cas des champs aléatoires conditionnels sur les séquences, l'algorithme d'inférence exacte que nous venons de définir est un cas particulier de l'algorithme *sum-product* sur l'arbre de jonction. En effet, on considère la figure 4.7. Celle-ci représente un exemple simple de graphe d'indépendances dans le modèles des 3-CRFs. Un arbre de jonction de ce graphe d'indépendances est fourni sur la figure 4.8. Plusieurs autres arbres de jonction étaient possibles, toutefois, celui que nous avons construit peut être généré automatiquement quel que soit le graphe d'indépendances dans le modèle des 3-CRFs. Sur cet arbre de jonction, on peut définir deux types de messages : les messages entre cliques voisines, c'est-à-dire ayant deux nœuds en commun comme c'est le cas de C_1 et C_2, et les messages d'une clique fille à une clique père, par exemple entre C_3 et C_1. On considère dans un premier temps le message transmis de C_3

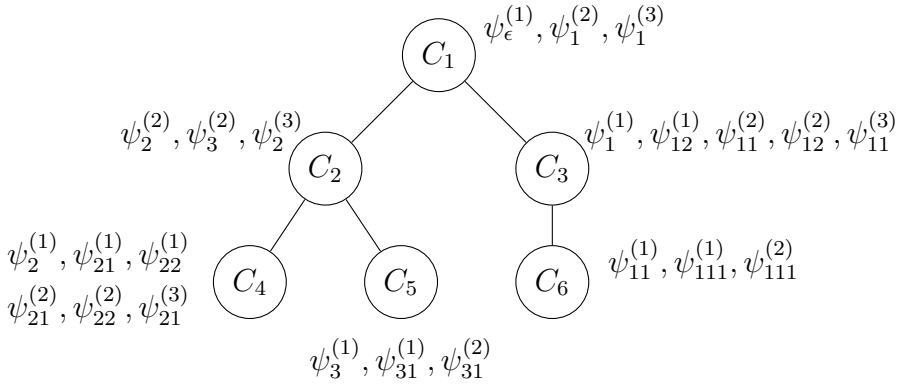

FIG. 4.8 – Arbre de jonction correspondant au graphe d'indépendances de la figure 4.7.

à C_1. Celui-ci s'exprime comme suit :

$$\begin{aligned}\mu_{C_3C_1}(y_1) &= \sum_{(y_{11},y_{12})\in\mathcal{Y}^2} \phi_{C_3}(y_1, y_{11}, y_{12})\mu_{C_6C_3}(y_{11}) \\ &= \sum_{(y_{11},y_{12})\in\mathcal{Y}^2} \psi_1^{(1)}\psi_{11}^{(2)}\psi_{12}^{(2)}\psi_{11}^{(3)}\mu_{C_6C_3}(y_{11}) \\ &= \beta_1(y_1)\end{aligned}$$

Ce message correspond à la variable $\beta_1(y_1)$. Dans le cas général, ce premier type de message correspond donc aux variables *inside* définies précédemment.

De la même façon, les messages entre deux cliques ayant deux nœuds en commun peuvent être définis sur cet arbre de jonction de la manière suivante :

$$\begin{aligned}\mu_{C_2C_1}(y_\epsilon, y_2) &= \sum_{y_3\in\mathcal{Y}} \phi_{C_2}(y_\epsilon, y_2, y_3)\mu_{C_4C_2}(y_2)\mu_{C_5C_2}(y_3) \\ &= \sum_{y_3\in\mathcal{Y}} \psi_2^{(2)}\psi_3^{(2)}\psi_2^{(3)}\beta_2(y_2)\beta_3(y_3) \\ &= \beta'_{\epsilon,2}(y_\epsilon, y_2)\end{aligned}$$

Ce deuxième type de message correspond donc aux variables *backward*. L'algorithme d'inférence que nous venons de décrire est donc bien un cas particulier de l'algorithme *sum-product* sur l'arbre de jonction.

4.2.2 Recherche du maximum a posteriori

Le problème de la recherche du maximum a posteriori, consiste à trouver l'annotation $\hat{y}$ la plus probable étant donnés une observation $\mathbf{x}$, un ensemble de fonctions de caractéristiques et leurs poids :

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}\in\mathcal{Y}^N} p(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}\in\mathcal{Y}^N} \left(\prod_n \psi_n^{(1)} \prod_{ni} \psi_{ni}^{(2)}\psi_{ni}^{(3)} \right)$$

Cette expression est très similaire à celle du coefficient de normalisation $Z(\mathbf{x})$, à la différence que la somme sur toutes les annotations possibles est remplacée par la fonction arg max. L'algorithme de recherche du maximum a posteriori est donc lui aussi très similaire à celui du calcul de $Z(\mathbf{x})$. On définit la variable $\delta_n(y_n)$ en remplaçant dans la définition de $\beta_n(y_n)$ la somme par la fonction max. Cette variable correspond ainsi à la probabilité non normalisée de l'annotation la plus probable du sous-arbre enraciné en n, où la racine est annotée par y_n.

$$\begin{aligned}
\text{Initialisation} &: \delta_n(y_n) = \psi_n^{(1)} \text{ si } n \text{ est une feuille } (ie.\ m = 0) \\
\text{Récursion} &: \delta_n(y_n) = \psi_n^{(1)} \max_{y_{n1} \in \mathcal{Y}} \delta'_{n,1}(y_n, y_{n1}) \\
\text{Fin} &: p(\hat{\mathbf{y}}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \max_{y_\epsilon \in \mathcal{Y}} \delta_\epsilon(y_\epsilon)
\end{aligned}$$

De la même façon, on adapte la variable $\beta'_{n.k}(y_n, y_{n.k})$ pour obtenir $\delta'_{n.k}(y_n, y_{n.k})$:

$$\begin{aligned}
\text{Initialisation} &: \delta'_{n,m}(y_n, y_{nm}) = \delta_{nm}(y_{nm})\psi_{nm}^{(2)} \\
\text{Récursion} &: \delta'_{n,k}(y_n, y_{nk}) = \delta_{nk}(y_{nk})\psi_{nk}^{(2)} \max_{y_{n(k+1)}} \left(\psi_{nk}^{(3)} \delta'_{n,k+1}(y_n, y_{n(k+1)})\right) \\
\text{Fin} &: \delta'_{n,1}(y_n, y_{n1}) = \max_{y_{n2},\ldots,y_{nm} \in \mathcal{Y}^m} \left(\prod_{i=1}^{m} \delta_{ni}(y_{ni}) \psi_{ni}^{(2)} \prod_{i=1}^{m-1} \psi_{ni}^{(3)}\right)
\end{aligned}$$

Ainsi, la probabilité de l'annotation la plus probable $\hat{\mathbf{y}}$ s'exprime de la façon suivante :

$$p(\hat{\mathbf{y}}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \max_{y_\epsilon} \delta_\epsilon(y_\epsilon) \tag{4.13}$$

L'annotation $\hat{\mathbf{y}}$ est obtenue en mémorisant, à chaque calcul de variable δ et δ' le label qui maximise la valeur de cette variable.

De la même façon que pour l'algorithme de calcul de $Z(\mathbf{x})$, on peut remarquer que l'algorithme de recherche du maximum a posteriori est en fait un cas particulier de l'algorithme *max-product* sur l'arbre de jonction. La complexité de cette algorithme est donc la même que pour le calcul de $Z(\mathbf{x})$: $\mathcal{O}(N \times |\mathcal{Y}|^3)$.

4.2.3 Estimation des paramètres

Nous abordons maintenant le problème de l'estimation des paramètres d'un champ aléatoire conditionnel. Ce problème consiste à trouver les poids Λ des fonctions de caractéristiques, étant donnés l'ensemble de ces fonctions et un ensemble S de couples $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ où $\mathbf{x}^{(i)}$ est une observation (un arbre XML) et $\mathbf{y}^{(i)}$ est son annotation. Comme dans le cas des champs aléatoires conditionnels sur les séquences, il existe d'autres méthodes d'estimation des poids, parmi lesquelles les méthodes de Monte Carlo [Neal, 1993]. Toutefois, nous abordons ici cette tâche avec le principe du maximum de vraisemblance.

Cette méthode consiste à dire que les poids optimaux sont ceux qui maximisent la vraisemblance des données, c'est-à-dire les couples de l'ensemble d'apprentissage, dans le

modèle. La fonction de vraisemblance conditionnelle s'exprime de la façon suivante :

$$L(\Lambda) = \prod_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) \tag{4.14}$$

où $p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})$ est la probabilité conditionnelle de $\mathbf{y}^{(i)}$ sachant $\mathbf{x}^{(i)}$ dans le champ aléatoire conditionnel avec les poids Λ. Afin de simplifier les calculs, nous utilisons la fonction de log-vraisemblance :

$$\mathcal{L}(\Lambda) = \sum_{(\mathbf{x}^{(i)},y^{(i)})\in S} \log p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) \tag{4.15}$$

Comme dans le cas des champs aléatoires conditionnels pour les séquences, il est nécessaire de pénaliser la log-vraisemblance de façon à éviter le surapprentissage. Toutefois, cette pénalisation ne modifiant en rien les algorithmes qui suivent, nous l'occultons dans un souci de lisibilité.

Cette fonction de log-vraisemblance possède la propriété intéressante d'être concave, ce qui garantit l'existence d'une unique solution optimale au problème de maximisation. Toutefois, cette solution ne peut pas être calculée analytiquement, car il est impossible de calculer la dérivée de cette fonction par rapport à l'ensemble des paramètres Λ. C'est pourquoi il est nécessaire d'utiliser des méthodes approchées. Pour maximiser la fonction de log-vraisemblance, de nombreuses méthodes ont été employées dans le cadre des champs aléatoires conditionnels sur les séquences, comme les méthodes dites d'*iterative scaling* [Lafferty et al., 2001] ou les méthodes à base de gradient. Une comparaison empirique de ces méthodes dans le cadre des champs aléatoires conditionnels [Wallach, 2003] a montré que les méthodes à base de gradient appelées méthodes de Newton, plus précisément l'algorithme des L-BFGS [Byrd et al., 1995], sont les plus rapides.

Afin de mettre en œuvre ces méthodes, il est nécessaire de calculer la dérivée partielle de la fonction de log-vraisemblance par rapport à chaque paramètre (ou poids) du champ aléatoire conditionnel. Cette dérivée par rapport au poids $\lambda_k^{(j)}$ d'une fonction définie sur une clique de taille j (1, 2 ou 3) s'exprime comme suit :

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_k^{(j)}} = \sum_{(\mathbf{x}^{(i)},\mathbf{y}^{(i)})\in S} \left[\sum_{c\in\mathcal{C}_j} f_k^{(j)}(\mathbf{y}_c^{(i)}, \mathbf{x}^{(i)}, c) - \Big(\sum_{\mathbf{y}\in\mathcal{Y}^N} p_\Lambda(\mathbf{y}|\mathbf{x}^{(i)}) \Big) \sum_{c\in\mathcal{C}_j} f_k^{(j)}(\mathbf{y}_c, \mathbf{x}^{(i)}, c) \right] \tag{4.16}$$

où $\mathcal{C}_j$ est l'ensemble des cliques de taille j dans le graphe d'indépendances.

Le calcul de cette dérivée comporte une somme sur l'ensemble des annotations possibles de chaque observation $\mathbf{x}^{(i)}$ de l'ensemble d'apprentissage S. Il est donc nécessaire de reformuler ce calcul de façon à éviter cette somme impliquant une complexité exponentielle en la taille N des arbres $\mathbf{x}^{(i)}$. Pour cela, on peut remarquer que cette dérivée peut se réécrire en utilisant les probabilités marginales :

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_k^{(j)}} = \sum_{(\mathbf{x}^{(i)},y^{(i)})\in S} \left[\sum_{c\in\mathcal{C}_j} f_k^{(j)}(\mathbf{y}_c^{(i)}, \mathbf{x}^{(i)}, c) - \sum_{c\in\mathcal{C}_j} \sum_{y_c} f_k^{(j)}(\mathbf{y}_c, \mathbf{x}^{(i)}, c) p_\Lambda(\mathbf{y}_c|\mathbf{x}^{(i)}) \right] \tag{4.17}$$

où $p_\Lambda(\mathbf{y}_c|\mathbf{x}^{(i)})$ est la probabilité marginale des labels affectés à la clique c de taille j. Un calcul efficace de la dérivée partielle de la fonction de log-vraisemblance passe donc par un calcul efficace des probabilités marginales.

Nous définissons donc un algorithme de programmation dynamique permettant de calculer les probabilités marginales s'inspirant à la fois des algorithmes *inside-outside* [Manning and Schütze, 1999], pour la nature hiérarchique des arbres, et *forward-backward* (*cf.* section 3.3.2.0), pour gérer l'arité non bornée des arbres XML. Pour cela, nous disposons déjà des variables *inside* $\beta_n(y_n)$ et *backward* $\beta'_{n,n.k}(y_n, y_{nk})$ définies pour le calcul de $Z(\mathbf{x})$. Nous introduisons maintenant deux variables supplémentaires. Tout d'abord, nous définissons une variable *outside* $\alpha_n(y_n)$ comme étant la somme des probabilités non-normalisées de toutes les annotations possibles du contexte du nœud en position n. Le **contexte** d'un nœud n d'un arbre est défini comme l'intégralité de l'arbre auquel on a enlevé le sous-arbre enraciné en n. La variable *outside* s'exprime comme suit :

$$\begin{aligned}
\text{Initialisation} &: \alpha_\epsilon(y_\epsilon) = 1 \\
\text{Récursion} &: \alpha_n(y_n) = \sum_{y_{n'} \in \mathcal{Y}} \alpha_{n'}(y') \frac{\beta_{n'}(y')}{\beta_n(y)} \text{ si } n' \text{ est le père de } n \\
\text{Fin} &: Z(\mathbf{x}) = \sum_{y_n \in \mathcal{Y}} \psi_n^{(1)} \alpha_n(y_n) \text{ si } n \text{ est une feuille}
\end{aligned}$$

Enfin, nous définissons les variables *forward* $\alpha'_{n,k}(y_n, y_{nk})$. Celles-ci sont semblables aux variables *backward* définies précédemment, si ce n'est qu'elles portent sur la partie gauche du sous-arbre enraciné en n au lieu de la partie droite. Elles s'expriment comme suit :

$$\begin{aligned}
\text{Initialisation} &: \alpha'_{n,1}(y_n, y_{n1}) = 1 \\
\text{Récursion} &: \alpha'_{n,k+1}(y_n, y_{n(k+1)}) = \sum_{y_{nk} \in \mathcal{Y}} \left(\beta_{nk}(y_{nk}) \psi_{nk}^{(2)} \psi_{nk}^{(3)} \alpha'_{n,k}(y_n, y_{nk}) \right) \\
\text{Fin} &: \alpha'_{n,m}(y_n, y_{nm}) = \sum_{y_{n1}, \ldots, y_{n(m-1)} \in \mathcal{Y}^{m-1}} \left(\prod_{i=1}^{m} \beta_{ni}(y_{ni}) \psi_{ni}^{(2)} \prod_{i=1}^{m-1} \psi_{ni}^{(3)} \right)
\end{aligned}$$

La figure 4.9 propose une représentation graphique des différentes variables de programmation dynamique. Elle montre sur quelles parties de l'arbre portent chacune de ces variables. On voit ainsi bien que la variable α_n correspond au contexte du nœud n, tandis que la variable β_n couvre le sous-arbre enraciné en n. Au sein de ce sous-arbre, les variables $\alpha'_{n,nk}$ et $\beta'_{n,nk}$ couvrent respectivement les parties précédant et suivant le nœud nk. Contrairement aux variables *inside* et *backward*, ces nouvelles variables ne sont pas des adaptations directes de l'algorithme *sum-product* sur l'arbre de jonction. En effet, elles ne correspondent pas directement à des messages passés entre deux cliques maximales.

A l'aide de ces nouvelles variables, il est maintenant aisé de calculer les probabilités marginales. Les équations 4.18 à 4.20 montrent comment ces probabilités marginales

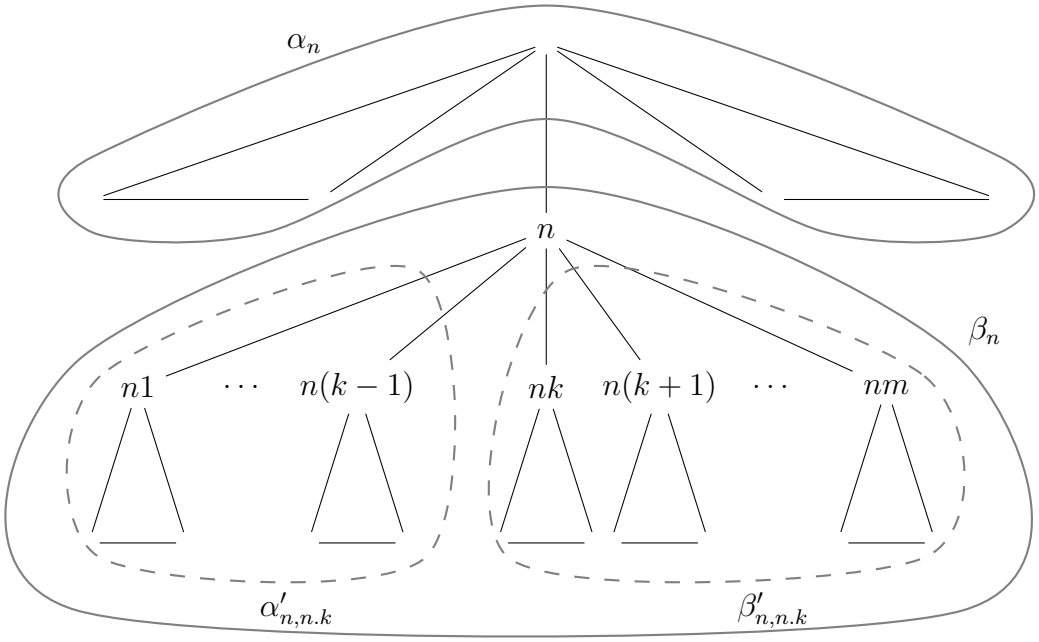

FIG. 4.9 – Schéma représentant les parties de l'arbre couvertes par les variables de programmation dynamique.

s'expriment avec ces variables :

$$p(y_n|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\alpha_n(y_n)\beta_n(y_n) \tag{4.18}$$

$$p(y_n, y_{ni}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\alpha_n(y_n)\psi_n^{(1)}\alpha'_{n,ni}(y_n, y_{ni})\beta'_{n,ni}(y_n, y_{ni}) \tag{4.19}$$

$$p(y_n, y_{ni}, y_{n(i+1)}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\alpha_n(y_n)\psi_n^{(1)}\psi_{ni}^{(2)}\psi_{ni}^{(3)}\alpha'_{n,ni}(y_n, y_{ni})\beta'_{n,n(i+1)}(y_n, y_{n(i+1)}) \tag{4.20}$$

Le calcul de la probabilité marginale d'une clique, que ce soit une clique à 1, 2 ou 3 nœuds, a donc la même complexité que dans les algorithmes précédents : celle-ci est linéaire dans la taille de l'arbre et cubique dans la taille de l'alphabet des labels. De l'équation (4.17), on déduit que le calcul d'une dérivée partielle a la même complexité. De plus, s'il est nécessaire de calculer, à chaque pas de gradient, la dérivée partielle par rapport à chaque paramètre du système, ces calculs n'augmentent pas la complexité car les probabilités marginales intervenant dans ces calculs sont mémorisées et leur calcul n'est donc effectué qu'une fois par pas de gradient. Le seul paramètre venant augmenter la complexité est le nombre de pas de gradient. La complexité de la tâche d'estimation des paramètres avec cet algorithme est donc $\mathcal{O}(N \times G \times |\mathcal{Y}|^3)$, où G est le nombre de pas de gradient.

4.2.4 Implémentation

Les 3 modèles de champs aléatoires conditionnels définis pour les arbres XML ainsi que les algorithmes d'inférence exacte et d'apprentissage précédemment décrits ont été

implantés par Hahn-Miss Tran dans un programme appelé XCRF[6] librement disponible. Nous décrivons ici divers aspects qui ont dû être pris en compte afin d'obtenir un paquet offrant des performances satisfaisantes en termes de temps de calcul.

Tout d'abord, ce programme a été implanté en Java[7]. Le choix de ce langage a été guidé essentiellement par la grande modularité qu'il offre, et par le nombre de librairies existantes qu'il propose. Ainsi, les représentations de matrices de la librairie COLT[8] ont pu être utilisées pour garantir un accès rapide aux variables de programmation dynamique. Aussi, le choix de l'algorithme de maximisation de la vraisemblance pour l'estimation des paramètres est aisément modifiable. Nous avons choisi d'utiliser l'implantation des L-BFGS proposée par le projet RISO[9].

Dans un premier temps, comme cela est souvent le cas pour l'implantation d'algorithmes sur des modèles exponentiels, l'intégralité des calculs a été effectué en passant au logarithme des valeurs calculées. Ainsi, non seulement moins d'exponentielles (présentes dans le calcul de chaque fonction de potentiel) sont calculées, mais surtout le passage au logarithme permet d'éviter la perte de précision rencontrée lors de calculs sur de très grands nombres à virgule flottante.

L'autre aspect à prendre en compte lors de l'implantation des algorithmes concerne la mémorisation des variables de programmation dynamique. En effet, pour que ces algorithmes soient efficaces, ces variables doivent être mémorisées afin de n'effectuer chaque calcul qu'une seule fois. Ainsi les variables *inside* et *outside* sont mémorisées dans des matrices de taille $N \times |\mathcal{Y}|$ où N est la taille de l'arbre, tandis que les variables *forward* et *backward*, qui prennent elles deux arguments, sont mémorisés dans des matrices de taille $N \times |\mathcal{Y}|^2$. L'utilisation de la librairie COLT garantit un accès rapide à ces matrices.

Enfin, sont aussi mémorisées les valeurs des fonctions de potentiel. Pour cela, on a trois matrices de taille $N \times |\mathcal{Y}|^i$, i variant de 1 à 3 selon le type de clique considéré. Toutefois, toutes les valeurs possibles de ces fonctions ne sont pas calculées. En effet, dans notre package, chacune de ces fonctions possède un test sur les annotations de la clique. Les valeurs de ces fonctions ne sont donc calculées que pour les cliques vérifiant ce test sur l'annotation. Cela réduit grandement le complexité car, dans de nombreux cas, ce test n'est satisfait que pour un nombre très restreint de cliques du graphes d'indépendances. Comme pour les variables de programmation dynamique, l'accès à ces matrices est optimisé en utilisant la librairie COLT.

4.3 Comparaison empirique des modèles

Nous allons maintenant comparer empiriquement sur des tâches d'annotation d'arbres les trois modèles de dépendances proposés précédemment. Le but de ces expériences est de valider l'intérêt de ces modèles selon les critères que sont :

- la capacité à apprendre à partir de peu d'exemples annotés
- le nombre de fonctions de caractéristiques nécessaires à l'obtention de bons résultats

[6] `http://treecrf.gforge.inria.fr`
[7] `http://java.sun.com`
[8] `http://dsd.lbl.gov/~hoschek/colt`
[9] `http://riso.sourceforge.net`

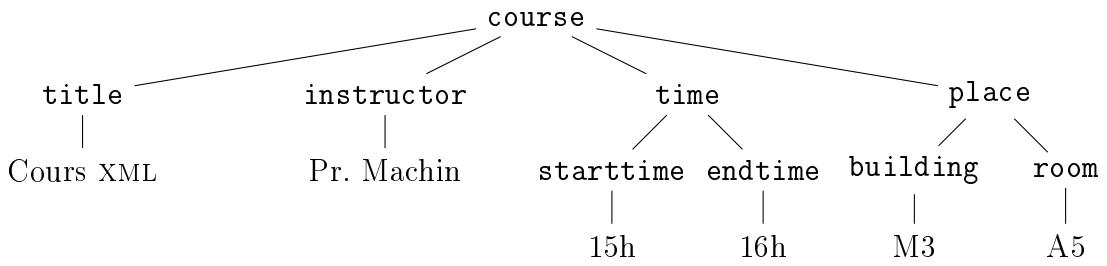

FIG. 4.10 – Exemple d'arbre XML du corpus "Courses".

– le temps nécessaire pour les phases d'apprentissage et d'annotation des arbres XML

Dans les expériences qui suivent, nous considérons l'existence d'une procédure de génération automatique des fonctions de caractéristiques. Cette procédure prend en paramètres un échantillon d'apprentissage constitué de couple (observation, annotation) et retourne un ensemble de fonction de caractéristiques définie sur les cliques à 1, 2 ou 3 nœuds. Cette procédure de génération est décrite en détails dans l'annexe A.

Nous présentons donc les résultats obtenus par nos trois modèles de champs aléatoires conditionnels pour les arbres XML sur une tâche d'annotation d'arbres sur deux corpus XML, ces deux corpus se distinguant essentiellement par la taille de l'alphabet des labels $\mathcal{Y}$ considéré.

4.3.1 Description des expériences

Les expériences que nous menons ici consistent en une même tâche sur deux corpus différents. Le but de cette tâche est d'annoter sémantiquement des arbres XML. Ainsi, à chaque nœud de l'arbre XML doit être attribué un label décrivant le sens de ce nœud dans l'arbre. Cette tâche peut être vue comme une forme très simple de transformation d'arbres où l'annotation est l'arbre de sortie. Dans cette transformation, la structure de l'arbre ne change pas, seules les étiquettes des nœuds sont changées. Pour cette tâche, nous utilisons dans un premier temps le corpus "Courses"[10] construit par Anhai Doan. Celui-ci est composé de 960 documents XML d'environ 20 nœuds chacun, ces documents représentant des descriptions de cours provenant de cinq universités américaines. La figure 4.10 montre un exemple de document XML de ce corpus. Nous effectuons aussi cette tâche d'annotation sur une sous-partie du corpus "Movie" de la tâche *Structure Mapping* du challenge XMLMining[11]. La sous-partie de ce corpus sur laquelle nous lançons les expériences est composée de 1081 documents XML d'environ 80 nœuds chacun. Chaque document XML contient la description détaillée d'un film (titre, réalisateur, synopsis, *etc.*). Cette première expérience sur ces deux corpus va nous servir à valider, sur des exemples où les dépendances entre labels sont fortes, l'intérêt croissant des différents modèles de dépendances et surtout le gain occasionné par la définition des cliques triangulaires.

Pour mettre en œuvre une tâche d'annotation sur ces deux corpus, nous avons choisi de remplacer les étiquettes de tous les nœuds de type élément des arbres XML de ces deux

[10] http://anhai.cs.uiuc.edu/archive
[11] http://xmlmining.lip6.fr

corpus par une unique étiquette `NODE` non informative, tandis que les feuilles texte sont laissées inchangées. La tâche d'annotation sémantique consiste alors à associer, à chaque nœud interne de l'arbre, l'étiquette qui lui était attribuée. L'alphabet des labels est alors l'ensemble des étiquettes possibles pour un élément dans la DTD de chaque corpus. Dans le cas du corpus "Courses", on compte 14 étiquettes différentes, tandis que le corpus "Movie" en compte 37. De plus, cette tâche d'annotation est d'autant plus difficile que les arbres d'entrée n'apportent que très peu d'informations. En effet, le choix des labels ne peut pas s'appuyer sur les étiquettes des nœuds des arbres d'entrée. De plus, cette tâche ne peut être résolue avec la simple connaissance de la DTD des arbres de sortie. En effet, on considère, dans le cas du corpus "Courses", la règle de la DTD concernant les éléments fils de la racine `course` :

```
<!ELEMENT course ((misc|title|credits|days|time|place|instructor)+,
                  (section|session)*)>
```

Cette règle indique que sous la racine, on peut avoir tout d'abord une succession de nœuds annotés par un label à choisir parmi sept (`misc`, `title`, *etc.*), puis éventuellement une succession de nœuds annotés par `section` et/ou `session`. La simple connaissance de cette règle ne permet effectivement pas d'annoter correctement les fils de la racine. La bonne réussite de cette tâche d'annotation dépend alors fortement de la capacité du modèle à utiliser la structure de l'arbre d'entrée ainsi que son contenu texte et à apprendre les dépendances entre les différents labels. Ainsi, sur l'exemple d'arbre XML du corpus "Courses" représenté sur la figure 4.10, le modèle va devoir apprendre par exemple qu'un nœud annoté par `time` doit avoir deux fils annotés par `starttime` et `endtime`. Il va aussi devoir apprendre, par exemple, que le nœud annoté par `starttime` possède une feuille texte dont le contenu est une heure. Ainsi, cette tâche possède une des difficultés majeures du domaine de l'annotation d'arbres : l'exploitation conjointe de la structure et du contenu textuel.

Dans les expériences suivantes, nous utilisons un échantillon d'apprentissage composé de L exemples annotés du corpus considéré pour apprendre un champ aléatoire conditionnel, tandis que le reste du corpus est utilisé pour évaluer les performances en termes d'annotation du champ aléatoire conditionnel appris. Selon les expériences, le nombre L d'exemples utilisés dans l'échantillon d'apprentissage varie entre 5, 10, 20 et 50. Afin de faire varier la quantité d'informations disponibles sur l'observation, on définit un paramètre que nous appelons paramètre de voisinage. Pour cela, on considère que chaque fonction de caractéristiques est constituée de la conjonction d'un test sur l'ensemble des labels de la clique considérée ainsi que d'un test sur l'observation $\mathbf{x}$, ce test pouvant porter sur des propriétés structurelles du nœud (profondeur, nombre de fils, *etc.*), ou sur son contenu textuel. Le paramètre de voisinage correspond alors à la distance maximale entre la clique considérée et le nœud de l'observation $\mathbf{x}$ sur lequel porte le test. Ainsi, plus ce paramètre est élevé, plus on dispose d'informations sur l'observation et plus le nombre de fonctions de caractéristiques est élevé. Pour plus de détails sur ce paramètre de voisinage, nous renvoyons à l'annexe A. Pour chaque taille de l'échantillon d'apprentissage, nous faisons varier ce paramètre de voisinage de 0 à 3. Enfin, avec ces différents jeux de paramètres, on compare les résultats obtenus par les trois modèles de dépendances pour les champs aléatoires conditionnels présentés précédemment. Les résultats présentés

Exemples	Voisinage	1-CRFs	2-CRFs	3-CRFs
5	0	42.92	69.46	**90**
	1	80.93	85.98	**88.07**
	2	79.35	**86.65**	84.76
	3	79.80	**89.51**	87.06
10	0	72.12	86.72	**91.87**
	1	88.29	91.08	**91.55**
	2	85.66	88.60	**94.51**
	3	88.25	88.20	**92.86**
20	0	45.50	78.77	**97.92**
	1	85.18	92.3	**97.17**
	2	90.43	95.88	**97.72**
	3	91.72	94.71	**99.97**
50	0	54.48	89.02	**98.56**
	1	92.95	96.25	**98.73**
	2	93.17	96.83	**99.26**
	3	93.39	96.89	**99.95**

TAB. 4.2 – Résultats pour la tâche d'annotation du corpus "Courses".

correspondent à la macro-moyenne de la F_1-mesure calculée sur chaque label des tâches d'annotation. Nous choisissons ce type de moyenne pour éviter le possible biais occasionné par la forte présence d'un label. Toutefois, bien que nous ne les présentions pas, les résultats avec la micro-moyenne sont très similaires à ceux présentés ci-après.

4.3.2 Résultats sur le corpus "Courses"

Les résultats de ces expériences sur le corpus "Courses" sont présentés dans le tableau 4.2. Plusieurs commentaires se dégagent de ces résultats. Tout d'abord, d'un point de vue général, il est intéressant de noter que, à deux exceptions près, quel que soit le nombre de documents en apprentissage et la valeur du paramètre de voisinage, les résultats obtenus avec les 2-CRFs sont nettement meilleurs que ceux des 1-CRFs, et les 3-CRFs obtiennent de meilleurs résultats que les 2-CRFs. Ce premier point montre que plus le modèle considère de dépendances entre les labels, meilleurs sont les résultats.

Une autre remarque concerne l'influence du nombre de documents dans l'échantillon d'apprentissage. En effet, on remarque d'une part que, quel que soit le modèle de dépendances considéré et la quantité d'informations disponible sur l'observation, les champs aléatoires conditionnels atteignent des résultats très proches de leur meilleur niveau dès l'utilisation de 20 exemples en apprentissage, soit seulement 2% du corpus. Ceci met bien en évidence l'intérêt d'un modèle tel que les champs aléatoires conditionnels. En effet, celui-ci étant un modèle conditionnel, il ne nécessite pas de modéliser l'observation (et toutes ses variations possibles), mais seulement les parties de l'observation utiles à une annotation correcte. Les résultats restent toutefois assez en retrait avec seulement 5 exemples en apprentissage. Ceci s'explique simplement par le fait qu'avec si peu d'exemples, certaines configurations de labels n'ont pas été rencontrées et il est donc impossible d'ap-

Exemples	1-CRFs		2-CRFs		3-CRFs	
	App.	Ann.	App.	Ann.	App.	Ann.
5	1.2	0.04	1.2	0.04	7.4	0.07
10	3.5	0.04	2.6	0.06	20.4	0.09
20	2.7	0.03	1.6	0.05	15.9	0.07
50	2.3	0.03	3.3	0.05	13.7	0.08

TAB. 4.3 – Temps de calcul (en secondes) pour la tâche d'annotation du corpus "Courses".

prendre à les annoter correctement.

Si l'on observe plus en détails les différences entre les différents modèles de dépendances choisis, on remarque deux éléments en faveur des 3-CRFs. D'une part, ils obtiennent, dès 10 exemples en apprentissage, des résultats supérieurs ou comparables à ceux des 1-CRFs et des 2-CRFs lorsque ceux-ci ont utilisé 50 exemples pour la phase d'apprentissage. Ceci illustre l'intérêt des cliques triangulaires introduites dans le graphe d'indépendances des 3-CRFs, qui permettent, grâce à l'utilisation importante des dépendances entre labels, d'apprendre un modèle performant à partir de peu d'exemples. Toutefois, cela peut être en partie dépendant de la tâche ici considérée. En effet, cette tâche possède des dépendances très fortes entre les labels, et les différences de performances que nous mesurons ici risquent d'être plus faibles sur d'autres tâches. L'autre résultat en faveur des 3-CRFs concerne la quantité d'informations sur l'observation nécessaire à l'obtention de bons résultats. En effet, dans le cas des 1-CRFs et des 2-CRFs, on observe de grosses différences de résultats selon que le paramètre de voisinage est réglé à 0 ou 3. Pour les 1-CRFs, lors de l'utilisation de 20 exemples en apprentissage, on voit les performances doubler en passant le paramètre de voisinage de 0 à 3. Dans le cas des 2-CRFs, on a un gain de près de 30% lors de l'utilisation de 5 exemples en apprentissage. À l'inverse, dans le cas des 3-CRFs, les résultats sont déjà très bons, et même parfois meilleurs, avec le paramètre de voisinage à 0 et le gain maximal observé est de 2% lorsque 20 exemples ont été utilisés en apprentissage.

L'autre critère que nous voulions évaluer est le temps de calcul que nécessite chaque modèle de dépendances afin de déterminer si le gain de qualité occasionné par la complexification du modèle de dépendances ne se fait pas au détriment du temps de calcul. Le tableau 4.3 présente, pour la phase d'apprentissage, le temps de calcul, en secondes, rapporté au nombre d'exemples et pour la phase d'annotation, le temps moyen d'annotation d'un document XML.

D'un point de vue général, les résultats confirment ce que la complexité des différents algorithmes laissait présager : les 1-CRFs sont les plus rapides et les 3-CRFs les plus lents. Toutefois, les différences sont bien moindres que celles annoncées par la complexité des algorithmes. En effet, la tâche d'annotation considérée comportant 14 labels, le passage d'un modèle à l'autre devrait entraîner une multiplication par 14 du temps de calcul. Pourtant, on remarque que les 2-CRFs sont plus ou moins aussi rapides que les 1-CRFs en apprentissage et moins de 2 fois plus lents en apprentissage. La différence est légèrement plus sensible entre les 2-CRFs et les 3-CRFs. En effet, l'apprentissage des 3-CRFs est en moyenne 6 fois plus lent que pour les 2-CRFs, tandis que l'annotation d'un document prend en moyenne 2 fois plus longtemps avec les 3-CRFs. Ces relativement faibles diffé-

Exemples	Voisinage	1-CRFs	2-CRFs	3-CRFs
5	0	64.71	97.44	**99.26**
	1	88.93	99.22	**99.33**
	2	98.84	98.36	**99.14**
	3	98.58	**98.82**	98.67
10	0	61.27	94.92	**98.66**
	1	89.87	98.61	**98.82**
	2	**98.68**	98.30	98.66
	3	**98.62**	98.55	98.43
20	0	62.34	**95.71**	95.18
	1	90.38	96.53	**96.54**
	2	98.17	98.19	**98.70**
	3	98.33	97.65	**98.70**
50	0	61.13	95.64	**99.62**
	1	91.43	98.82	**99.55**
	2	**99.85**	99.60	99.65
	3	**99.87**	99.66	99.72

TAB. 4.4 – Résultats pour la tâche d'annotation du corpus "Movie".

rences de temps de calcul ont essentiellement deux causes. D'une part, l'optimisation de l'implantation de l'algorithme a permis de réduire la différence de complexité théorique en ne calculant que les valeurs nécessaires, comme expliqué dans la section 4.2.4 sur l'implantation. De plus, lors de la phase d'apprentissage des paramètres, on remarque souvent que, lorsque le modèle de dépendances est plus complexe, la descente de gradient converge plus vite, et le nombre d'étapes de gradient est donc plus faible.

Enfin, la variation du nombre d'étapes de la descente de gradient est aussi responsable des variations du temps d'apprentissage rapporté au nombre d'exemples. En effet, la complexité de l'algorithme d'apprentissage est linéaire dans la taille de l'échantillon d'apprentissage.

4.3.3 Résultats sur le corpus "Movie"

Le tableau 4.4 présente les résultats de la tâche d'annotation du corpus "Movie". On remarque sur ce second corpus que les différences de performance entre les trois modèles de dépendances pour les champs aléatoires conditionnels sont moins nettes. Ceci s'explique par le fait que le corpus "Movie" est légèrement plus simple que le corpus "Courses". En effet, la DTD du corpus "Movie" est beaucoup plus contrainte, c'est-à-dire qu'elle reconnaît beaucoup moins de différences de structures entre les différents arbres qui respectent cette DTD. Par exemple, dans le corpus "Courses", l'ordre des fils d'un élément était très variable d'un document XML à l'autre, et le bon ordre devait ainsi être retrouvé par les champs aléatoires conditionnels lors de l'annotation. Dans le cas du corpus "Movie", l'ordre des fils d'un élément est beaucoup moins variable. Par exemple, le premier fils d'un élément `movie` est toujours un élément `presentation`.

On note toutefois que, malgré une difficulté moindre, la tendance générale des résultats

Exemples	1-CRFs		2-CRFs		3-CRFs	
	App.	Ann.	App.	Ann.	App.	Ann.
5	72	1.84	59.4	1.56	482.4	2.28
10	32.4	1.49	36.2	1.66	286.9	2.26
20	53.3	1.52	67.4	1.78	537.6	2.02
50	60.82	1.73	56.46	1.87	592.38	2.78

TAB. 4.5 – Temps de calcul (en secondes) pour la tâche d'annotation du corpus "Movie".

obtenus sur le corpus "Courses" se maintient. Tout d'abord, dans l'ensemble, les meilleurs résultats sont une fois de plus obtenus avec les 3-CRFs, même si le meilleur score est obtenu par les 1-CRFs avec 50 documents en apprentissage et le paramètre de voisinage à 3. De plus, les 2-CRFs et 3-CRFs nécessitent moins d'informations sur l'observation que les 1-CRFs pour obtenir de bon résultats. En effet, quelle que soit la valeur du paramètre de voisinage, les 2-CRFs et les 3-CRFs obtiennent systématiquement une F_1-mesure supérieure à 90%. Enfin, on remarque que sur ce corpus, le nombre d'exemples en apprentissage n'influe que très peu la qualité du champ aléatoire conditionnel et les résultats sont très stables quel que soit le nombre d'exemples utilisés, surtout dans le cas des 2-CRFs et des 3-CRFs. Ceci montre la capacité des champs aléatoires conditionnels à apprendre à annoter des arbres XML correctement à partir de peu d'exemples.

Le tableau 4.5 présente les temps de calcul pour l'apprentissage et l'annotation sur le corpus "Movie". On remarque que, contrairement aux expériences sur le corpus "Courses", les temps de calcul sont ici prohibitifs. En effet, si les temps d'annotation, bien que dépassant 1 seconde par document, restent raisonnables quel que soit le modèle de dépendances utilisé, les temps d'apprentissage sont quant à eux bien plus élevés, et rendent les champs aléatoires conditionnels, et plus particulièrement les 3-CRFs, inutilisables dans le cadre d'applications interactives (par exemple sur le Web). En effet, on atteint, dans le cas des 3-CRFs avec 50 exemples, un temps d'apprentissage de 592 secondes par exemple, soit plus de 8 heures pour l'ensemble de l'apprentissage. Cela met en évidence le principal défaut des 2-CRFs et surtout des 3-CRFs : lorsque l'alphabet des labels est trop grand, leur utilisation dans une application réelle devient impossible. Il devient donc nécessaire d'apporter aux champs aléatoires conditionnels, pour l'annotation d'arbres ainsi que pour l'annotation d'autres types de données, des méthodes permettant de réduire drastiquement la complexité. Pour cela, nous avons d'une part la possibilité d'utiliser les algorithmes d'inférence approchée, parmi lesquelles les méthodes variationnelles [Jordan et al., 1999], ou encore l'algorithme appelé *loopy belief progation* [Murphy et al., 1999]. Toutefois, si ces méthodes ont prouvé leur qualité et leur capacité à réduire grandement la complexité des algorithmes d'inférence pour les modèles graphiques, elles possèdent, à notre avis, le désavantage de ne pas tenir compte des connaissances du domaine pour réduire la complexité. Nous proposons donc, dans le chapitre suivant, deux méthodes permettant de réduire la complexité des algorithmes d'inférence exacte et d'apprentissage pour les champs aléatoires conditionnels, tout en utilisant des connaissances du domaine.

Chapitre 5

Optimisation des champs aléatoires conditionnels

Nous l'avons remarqué dans le chapitre précédent : le modèle des 3-CRFs, bien qu'offrant les meilleurs résultats pour annoter des arbres XML, possède des limitations du point de vue de la complexité des algorithmes d'annotation et d'apprentissage. Mais ce problème de lenteur des champs aléatoires conditionnels, essentiellement concernant l'algorithme d'apprentissage, se retrouve dans de nombreux autres domaines que celui de l'annotation d'arbres, notamment l'annotation de séquences ou l'utilisation des champs aléatoires conditionnels pour le traitement de l'image. L'origine de ce problème est le facteur $|\mathcal{Y}|^K$, où K est la taille de la clique maximale du graphe d'indépendances, que l'on trouve dans la complexité des algorithmes d'inférence exacte et d'apprentissage. En effet, il est nécessaire, lors de l'utilisation d'algorithmes d'inférence exacte et d'apprentissage, de considérer, pour chaque clique du graphe, l'ensemble des combinaisons de labels possibles et de calculer les fonctions de potentiel pour chacune de ces combinaisons.

Pour réduire ce problème de complexité, une première solution non satisfaisante consiste à réduire les dépendances du modèle. Toutefois, comme les expériences du chapitre précédent le montrent, les dépendances entre les variables aléatoires correspondant à l'annotation sont souvent ce qui permet au modèle d'obtenir de bons résultats. Nous proposons donc deux solutions à ce problème. D'une part, nous définissons des contraintes sur les affectations de labels possibles au niveau des cliques afin de réduire le nombre d'annotations possibles d'un arbre. D'autre part, nous proposons trois méthodes de composition de champs aléatoires conditionnels qui permettent d'approximer un CRF défini sur l'ensemble de l'alphabet des labels $\mathcal{Y}$ par plusieurs CRFs définis sur des sous-partie de $\mathcal{Y}$. Ces méthodes réduisent ainsi la complexité en diminuant le nombre de labels considérés par CRF. Dans la suite du chapitre, nous illustrons ces deux méthodes sur le modèle des 3-CRFs. Toutefois, celles-ci peuvent s'appliquer de la même façon à tous types de champs aléatoires conditionnels.

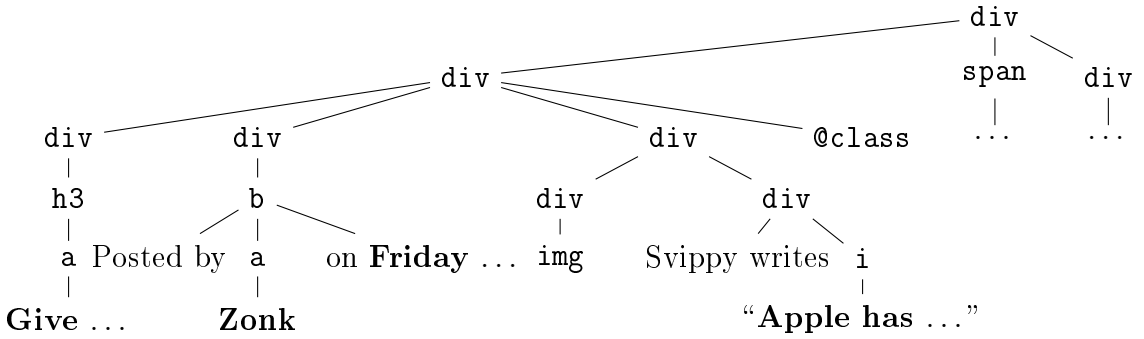

FIG. 5.1 – Arbre XHTML d'une page Web du site *Slashdot*.

5.1 Introduction de contraintes

Nous commençons par présenter l'introduction de contraintes dans les champs aléatoires conditionnels. Le principe de ces contraintes est de réduire, au niveau des cliques du graphe d'indépendances, le nombre de combinaisons de labels possibles afin de réduire la complexité due au facteur $|\mathcal{Y}|^k$. Nous allons, dans un premier temps, définir en détails les contraintes que nous introduisons dans les champs aléatoires conditionnels, et discuter leur expressivité. Ensuite, nous ferons un rapide tour d'horizon des travaux existants dans ce domaine, avant de terminer par une série d'expériences montrant l'intérêt des contraintes en termes de complexité et de qualité des résultats obtenus.

5.1.1 Définition des contraintes

Nous définissons maintenant les contraintes pour les champs aléatoires conditionnels, quel que soit le modèle de dépendances considéré. Le but des contraintes est de réduire le nombre de combinaisons de labels possibles au niveau des cliques du graphe d'indépendances en identifiant des combinaisons qui ne sont pas réalisables. L'introduction de contraintes signale au champ aléatoire conditionnel de ne pas considérer ces configurations lors de la recherche du maximum a posteriori ou lors du calcul des probabilités marginales et permet de réduire drastiquement la complexité de ces algorithmes. L'identification de ces combinaisons est une forme d'utilisation de connaissances du domaine. En effet, ces connaissances peuvent être d'origines diverses : sémantique des labels (dans les tâches d'annotation *Part-Of-Speech* sur les séquences), schéma des arbres de sortie, *etc.*

Considérons par exemple la tâche de transformation d'arbres décrite dans la section 1.2.1 et dont les figures 5.1 et 5.2 représentent un arbre d'entrée et son annotation de type "opérations d'édition". Si l'on considère l'ensemble de la DTD de RSS 2.0, un maximum de 63 labels différents sont possibles dans l'alphabet $\mathcal{Y}$, soit, au niveau des cliques triangulaires, un total d'environ $63^3 \approx 250000$ combinaisons de labels possibles. Ce nombre est naturellement prohibitif. Toutefois, en connaissant la DTD de RSS 2.0, ou tout simplement en observant l'annotation de la figure 5.2, on peut observer que certaines combinaisons de labels sont impossibles. Par exemple, on peut remarquer que tous les fils d'un nœud annoté par `delST` sont eux aussi annotés par `delST`, quelle que soit leur position dans l'arbre. Dans ce premier cas, c'est de la sémantique des labels (`delST` signifie "supprimer

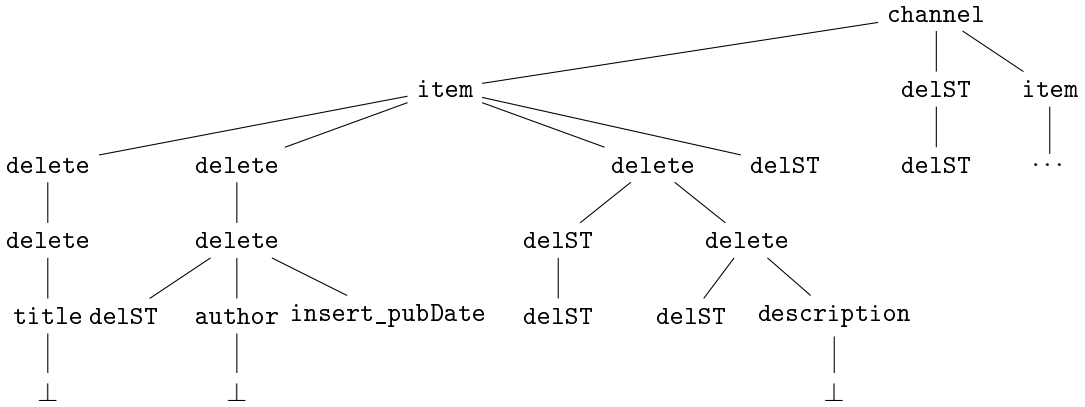

FIG. 5.2 – Annotation de l'arbre XHTML de la figure 5.1.

le sous-arbre") que provient la connaissance. De la même façon, un nœud annoté par `item` ne peut jamais apparaître comme fils d'un nœud annoté par `title`. En effet, d'après la DTD de RSS, un élément `title` ne peut avoir de fils `item`. Enfin, une feuille texte est, dans notre exemple, nécessairement annotée par un label parmi $\{\bot, \texttt{delST}, \texttt{insert\_pubDate}\}$.

On remarque que, parmi les connaissances ci-dessus, certaines permettent d'éliminer des combinaisons de labels, quelle que soit la position dans le graphe d'indépendances, tandis que d'autres ne s'appliquent qu'à certains endroits précis vérifiant un test sur l'observation. Pour modéliser ce type de connaissances sur l'annotation, nous définissons des contraintes.

Définition 5.1 *Une **contrainte** est une combinaison de labels interdite pour un type de clique t donné, conditionnée par un test portant sur l'observation* $\mathbf{x}$*. Cette contrainte s'applique sur toutes les cliques de type t pour lesquelles le test sur l'observation est satisfait.*

De plus, on note $\mathcal{S}_c$ l'ensemble des combinaisons de labels autorisées pour la clique c, c'est-à-dire les combinaisons de labels qui ne violent pas les contraintes. $\mathcal{S}_c$ est inclus dans l'ensemble de toutes les combinaisons de labels de $\mathcal{Y}$ possibles : $\mathcal{S}_c \subset \mathcal{Y}^{|c|}$. Ainsi, pour l'exemple d'annotation de la figure 5.2, on peut alors définir les contraintes stipulant qu'une feuille est nécessairement annotée par un label parmi $\bot$, `delST` ou `insert_pubDate` de la manière suivante :

$$\forall n \in \mathcal{C}_1,\ feuille(n) \Rightarrow \forall y \in \mathcal{Y} \setminus \{\bot, \texttt{delST}, \texttt{insert\_pubDate}\},\ y \notin \mathcal{S}_n \qquad (5.1)$$

où $\mathcal{C}_1$ est l'ensemble des cliques à 1 nœud et $feuille(n)$ est un prédicat qui retourne *vrai* si le nœud n est une feuille texte.

Certaines contraintes particulières s'appliquent sur toutes les cliques de même type t quelle que soit la position de la clique dans le graphe d'indépendances. C'est le cas, dans l'exemple de la figure 5.2, des contraintes stipulant que les fils d'un nœud annoté par `delST` sont eux aussi annotés par `delST`. De telles contraintes peuvent être exprimées comme suit :

$$\forall c \in \mathcal{C}_2\ \forall y \in \mathcal{Y} \setminus \{\texttt{delST}\},\ (\texttt{delST}, y) \notin \mathcal{S}_c \qquad (5.2)$$

où $\mathcal{C}_2$ est l'ensemble des cliques composées de deux nœuds dans le modèle des 3-CRFs. Ces contraintes, que nous appelons **contraintes uniformes**, sont un cas particulier des contraintes définies précédemment dans lesquelles le test sur l'observation $\mathbf{x}$ retourne toujours *vrai*.

Ces contraintes étant définies au niveau des cliques du graphe d'indépendances, elles ne changent pas le modèle de dépendances choisi. En effet, de telles contraintes peuvent être définies sous forme de fonctions de caractéristiques. Par exemple, dans le cas des 3-CRFs, la contrainte 5.1 est représentée par la fonction suivante :

$$f(y_n, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n \in \{\perp, \texttt{delST}, \texttt{insert\_pubDate}\} \\ & \text{et si le nœud } n \text{ est une feuille texte.} \\ 0 & \text{sinon.} \end{cases}$$

En associant à cette fonction de caractéristiques un poids de $-\infty$, on a donc, d'après la définition des fonctions de potentiel sur les cliques à 1 nœud (*cf.* équation (4.1)), une fonction de potentiel $\psi_n^{(1)}(y_n, \mathbf{x})$ valant 0 à la position n. Ainsi, toutes les annotations $\mathbf{y}$ pour lesquelles cette fonction s'active au moins une fois ont une probabilité nulle. Ceci assure que les seules annotations dont la probabilité n'est pas nulle sont celles qui respectent les contraintes.

5.1.2 Probabilité et algorithmes pour les champs aléatoires conditionnels avec contraintes

Avec les contraintes ainsi définies, la probabilité jointe $p(\mathbf{y}|\mathbf{x})$ d'une annotation $\mathbf{y}$ sachant l'observation $\mathbf{x}$ dans un champ aléatoire conditionnel, pour un graphe d'indépendance quelconque, s'exprime comme suit :

$$p(\mathbf{y}|\mathbf{x}) = \begin{cases} \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c, \mathbf{x}) & \text{si Sat}(\mathbf{y}) \\ 0 & \text{sinon} \end{cases} \tag{5.3}$$

où $\text{Sat}(\mathbf{y})$ est vrai si et seulement si $\mathbf{y}$ ne viole aucune contrainte. Ainsi, quand l'annotation $\mathbf{y}$ respecte les contraintes, le calcul de la probabilité conditionnelle ne change pas, à l'exception du coefficient de normalisation qui lui devient :

$$Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}^N \text{ tel que Sat}(\mathbf{y})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c, \mathbf{x}) \tag{5.4}$$

où N est la taille du champ aléatoire $\mathbf{Y}$. Au lieu de sommer sur toutes les annotations possibles, $Z(\mathbf{x})$ devient une somme sur toutes les annotations respectant les contraintes définies.

L'introduction de contraintes dans les champs conditionnels aléatoires vient aussi modifier les algorithmes d'inférence exacte et d'apprentissage. Étant donné que les contraintes que nous avons définies peuvent être utilisées quel que soit le graphe d'indépendances (*ie.* pas uniquement dans le cas des champs aléatoires conditionnels sur les arbres XML), nous présentons ici les modifications de ces algorithmes en termes de modifications des messages pour les algorithmes *sum-product* et *max-product* sur l'arbre de jonctions (*cf.* section 3.1.5). La modification de l'algorithme spécifique aux 3-CRFs est sur le même modèle.

L'utilisation des contraintes dans la recherche du maximum a posteriori paraît la plus naturelle. En effet, elle consiste à ne pas considérer, dans l'espace de recherche, les annotations violant des contraintes et dont on sait a priori que leur probabilité est nulle. Il en résulte, dans l'algorithme *max-product*, que lors du calcul des messages $\delta_{c'c}(\mathbf{y}_{c\cap c'}, \mathbf{x})$ entre deux cliques c et c', le message n'est pas calculé pour les valeurs de $\mathbf{y}_{c\cap c'}$ qui violent les contraintes. De plus, quand ce message est calculé, la fonction max ne s'applique que pour les combinaisons de labels $\mathbf{y}_{c'\setminus c}$ qui satisfont les contraintes, au lieu de s'appliquer sur l'ensemble des valeurs possibles. Ainsi, le message de l'équation (3.10) devient :

$$\delta_{c'c}(\mathbf{y}_{c\cap c'}, \mathbf{x}) = \max_{\mathrm{Sat}(\mathbf{y}_{c'\setminus c})} \phi_{c'}(\mathbf{y}_{c'}, \mathbf{x}) \prod_{c''\in\mathcal{N}(c')\setminus c} \delta_{c''c'}(\mathbf{y}_{c''\cap c'}, \mathbf{x}) \tag{5.5}$$

où $\mathrm{Sat}(\mathbf{y}_c)$ correspond à l'ensemble des combinaisons de labels autorisées par les contraintes pour la clique c.

Lors de l'apprentissage, il est nécessaire à la fois de calculer le coefficient de normalisation $Z(\mathbf{x})$ et les probabilités marginales, en tenant compte des contraintes. Pour cela, on adapte l'algorithme *sum-product* de la même façon que l'algorithme *max-product*. En effet, il suffit de remplacer dans la définition des messages $\mu_{c'c}(\mathbf{y}_{c\cap c'}, \mathbf{x})$ la somme sur tous les labels possibles par une somme sur tous les labels qui satisfont les contraintes. Le message de l'équation (3.7) devient :

$$\mu_{c'c}(\mathbf{y}_{c\cap c'}, \mathbf{x}) = \sum_{\mathrm{Sat}(\mathbf{y}_{c'\setminus c})} \phi_{c'}(\mathbf{y}_{c'}, \mathbf{x}) \prod_{c''\in\mathcal{N}(c')\setminus c} \mu_{c''c'}(\mathbf{y}_{c''\cap c'}, \mathbf{x}) \tag{5.6}$$

De plus, ces messages ne sont pas calculés pour les valeurs de $\mathbf{y}_{c\cap c'}$ qui violent les contraintes.

L'introduction des contraintes ne change pas l'expression de la probabilité $p(\mathbf{y}|\mathbf{x})$. Celle-ci s'exprime en effet toujours sous la forme d'un produit, normalisé par un coefficient $Z(\mathbf{x})$, de fonctions de potentiel correspondant à l'exponentielle d'une somme pondérée de fonctions de caractéristiques. Ainsi, comme pour les champs aléatoires conditionnels dans le cas général, la fonction de log-vraisemblance est concave et il existe une unique solution optimale au problème de l'apprentissage des poids des fonctions de caractéristiques d'un champ aléatoire conditionnel.

5.1.3 Intérêt et limitations des contraintes

L'intérêt d'intégrer des contraintes dans les champs aléatoires conditionnels est double. D'une part, l'intérêt principal est le gain en complexité. En effet, au lieu de calculer les fonctions de potentiel de chaque clique pour toutes les combinaisons de labels possibles, celles-ci ne sont calculées que pour les combinaisons de labels ne violant pas les contraintes. Ainsi, la complexité en temps de calcul passe de $\mathcal{O}(N \times |\mathcal{Y}|^k)$ à $\mathcal{O}(N \times M)$, où M correspondant à la taille du plus grand grand ensemble $\mathcal{S}_c$ de combinaisons de labels possibles pour toutes les cliques c du graphe d'indépendances :

$$M = \max_{c\in\mathcal{C}} |\mathcal{S}_c|$$

Si cette nouvelle complexité paraît linéaire, il faut tout de même remarquer que dans le pire des cas, c'est-à-dire quand aucune contrainte n'est définie, $M = |\mathcal{Y}|^k$ ou k est la taille des cliques maximales du graphe d'indépendances. Toutefois, quand de nombreuses contraintes peuvent être définies, M est largement plus petit que $|\mathcal{Y}|^k$. Ainsi, pour l'exemple de la tâche de transformation de données du format XHTML au format RSS de la figure 5.2, on a 7 labels possibles. En théorie, pour les cliques triangulaires, un total de $7^3 = 343$ combinaisons de labels possibles. L'introduction de contraintes liées à la DTD ou à la sémantique des labels telles que celles décrites précédemment, le nombre de combinaisons de labels possibles est réduit à 104, impliquant une complexité 3 fois inférieure.

L'autre intérêt des contraintes dans les champs aléatoires conditionnels concerne l'expressivité. En effet, deux champs aléatoires conditionnels possédant les mêmes fonctions de caractéristiques avec les mêmes poids, l'un étant contraint et l'autre non, ne fourniront pas nécessairement la même annotation pour une observation $\mathbf{x}$ donnée. Si un champ aléatoire conditionnel sans contraintes ne possède pas assez de fonctions de caractéristiques pour faire le bon choix d'annotation pour une variable aléatoire, une contrainte peut venir corriger cette erreur, ou à défaut de la corriger, la modifier. Par exemple, dans la tâche de transformation d'arbres XHTML en arbres RSS, nous considérons deux champs aléatoires conditionnels CRF_1 et CRF_2 avec les mêmes fonctions de caractéristiques et les mêmes paramètres. Aucune contrainte n'est définie pour le premier, tandis que le second possède les contraintes (5.1) et (5.2). Si CRF_1 fait l'erreur d'affecter le label `title` à un nœud fils d'un nœud annoté par `delST`, cette erreur ne pourra être faite par CRF_2 en raison de la contrainte (5.2). De la même façon, là où CRF_1 peut faire l'erreur d'affecter un autre label que $\perp$, `delST` ou `insert_pubDate` à une feuille, la contrainte (5.1) empêche CRF_2 de faire cette erreur. Toutefois, lorsque le champ aléatoire conditionnel sur lequel aucune contrainte n'est définie possède suffisamment de fonctions de caractéristiques pour modéliser la distribution de probabilité correcte, l'ajout de contraintes ne modifie pas cette distribution de probabilité et l'intérêt des contraintes se résume au gain de complexité.

Les contraintes que nous avons définies dans cette section possèdent toutefois quelques limitations. En effet, celles-ci doivent préserver la notion de séparabilité des calculs, c'est-à-dire conserver la possibilité d'effectuer des calculs indépendants pour chaque clique du graphe d'indépendances. En préservant ce principe, on est assuré de ne pas augmenter la complexité. C'est pour cela que les contraintes sont définies au niveau des cliques. Certains types de contraintes sont donc impossibles à définir. Par exemple, les contraintes sur le nombre d'occurrences d'un label du type $occ(l) \geq k$ ou $occ(l) \leq k$, où $k \in \mathbb{N}$ et $occ(l)$ est le nombre d'occurrences du label l dans l'annotation, ne respectent pas la factorisation en cliques. En introduisant ce type de contraintes, on introduit aussi des dépendances entre toutes les variables aléatoires pouvant prendre la valeur l. On obtiendrait ainsi un graphe d'indépendances complètement connecté, résultant en des algorithmes d'inférence exacte et d'apprentissage de complexité exponentielle en la taille N du graphe d'indépendances. De la même façon, dans le cas des champs aléatoires conditionnels pour les séquences, une contrainte du type "le label l_1 ne peut pas apparaître après le label l_2", ou "le label l_1 ne peut pas apparaître sous le label l_2" dans le cas des 3-CRFs, sont impossibles à exprimer. En effet, pour pouvoir les introduire, il faudrait une fois encore avoir un graphe d'indépendances complètement connecté.

Dans le cas des champs conditionnels aléatoires sur les arbres XML, certaines contraintes ressemblant à des contraintes longues distances, c'est-à-dire des contraintes concernant plusieurs nœuds n'apparaissant pas dans une même clique, peuvent toutefois être exprimées à l'aide de contraintes uniformes définies au niveau des cliques. Par exemple, dans le cas des 3-CRFs, la contrainte suivante :

$$\forall c \in \mathcal{C}_3 \; \forall (y_1, y_2) \in \mathcal{Y}^2 \setminus \{(\mathtt{a}, \mathtt{a})\}, \; (\mathtt{a}, y_1, y_2) \notin \mathcal{S}_c$$

implique que dans une clique triangulaire dont le père est annoté par a, les fils sont aussi annotés par a. Toutefois, par récurrence, il en résulte que tous les nœuds du sous-arbre enraciné en un nœud annoté par a sont eux aussi annotés par a. L'inverse, c'est-à-dire "tous les nœuds du sous-arbre ne sont pas annotés par a", n'est par contre pas exprimable. Par exemple, les contraintes que nous avons définies ne permettent pas d'exprimer le fait que dans un sous-arbre dont la racine est annotée par a il ne peut y avoir aucun nœud annoté par a.

Enfin, le dernier problème concernant les contraintes concerne l'implémentation. En effet, dans le cas général, on peut avoir, étant donnés un alphabet des labels $\mathcal{Y}$ et un arbre d'entrée $\mathbf{x}$ à annoter, un ensemble $\mathcal{S}_c$ de combinaisons de labels possibles différent pour chaque clique c du graphe d'indépendances. Ceci pose un problème de représentation et peut rapidement occuper une grande quantité de mémoire. Toutefois, on remarque que dans la pratique, de nombreuses cliques ont exactement le même ensemble de combinaisons de labels possibles. Il est donc possible de créer un type pour chaque ensemble de combinaisons de labels possibles et d'ensuite affecter ces types aux cliques. La création de ces types nécessite néanmoins un calcul supplémentaire et peut légèrement affecter le gain de complexité occasionné par les contraintes.

5.1.4 Travaux existants

Les travaux de [Kristjansson et al., 2004] ont proposé d'intégrer des contraintes dans les champs aléatoires conditionnels. Ceux-ci se placent dans le cadre de l'extraction d'informations, plus particulièrement pour les systèmes d'extraction d'informations interactive. De tels systèmes présentent à un utilisateur, dans une interface, les premiers résultats d'une extraction. L'utilisateur peut alors soit valider les résultats obtenus par le système, soit les corriger en spécifiant manuellement l'annotation d'un élément. Par exemple, sur la page Web issue de *Slashdot* de la figure 1.7, l'utilisateur peut ainsi spécifier que le titre est "Give iPod Thieves an Unchargeable Brick". Ces annotations faites manuellement par l'utilisateur sont alors ajoutées sous la forme de contraintes du type : "le label à la position i est l_i" ou "le label à la position i n'est pas l_i". L'article propose ensuite une adaptation de l'algorithme de Viterbi afin de permettre la recherche du maximum a posteriori vérifiant ces contraintes et de l'algorithme *forward-backward* pour calculer le coefficient de normalisation $Z(\mathbf{x})$.

D'une part, les contraintes définies dans cet article correspondent à un cas particulier de celles que nous venons de définir. En effet, dans ces travaux, le test sur l'observation est limité à un test sur la position i considérée, tandis que les contraintes que nous avons définies permettent de d'effectuer des tests sur l'intégralité de l'observation. De plus, les

```
<!ELEMENT time (starttime,endtime)>
<!ELEMENT starttime (#PCDATA)>
<!ELEMENT endtime (#PCDATA)>
<!ELEMENT place (building,room)>
<!ELEMENT building (#PCDATA)>
<!ELEMENT room (#PCDATA)>
```

FIG. 5.3 – Extrait de la DTD de sortie du corpus "Courses".

algorithmes d'inférence exacte avec ces contraintes n'ont été adaptés que dans le cas des champs aléatoires conditionnels sur les séquences. Enfin, alors que nous pensons que l'intégration des contraintes est intéressante à la fois lors de l'annotation et l'apprentissage des paramètres, ces travaux les mettent en œuvre uniquement pour l'annotation. Toutefois, cette dernière limitation est due au cadre applicatif de l'extraction d'informations interactive dans lequel se placent ces travaux.

5.1.5 Expériences avec contraintes

Nous évaluons maintenant expérimentalement l'intérêt des contraintes sur diverses tâches d'annotations d'arbres XML. Ces expériences nous permettent de mettre en évidence plusieurs points. Dans un premier temps, nous évaluons le gain de complexité apporté par les contraintes en mesurant le temps de calcul nécessaire à l'apprentissage des paramètres du champ aléatoire conditionnel et le temps d'annotation moyen d'un document XML du corpus. De plus, les contraintes permettant d'éliminer des configurations de labels possibles, nous mesurons aussi la qualité de l'annotation obtenue avec la précision, le rappel et la F_1-mesure au niveau des labels.

Description des expériences

Dans le cadre de nos expériences, nous considérons les tâches d'annotation des arbres XML des corpus "Courses" et "Movie" utilisés précédemment (*cf.* section 4.3.1) pour évaluer les champs aléatoires conditionnels sur les arbres XML. Nous comparons les résultats obtenus sur ces deux tâches avec les champs aléatoires conditionnels contraints à ceux obtenus avec les champs aléatoires conditionnels sans contraintes présentés dans le tableau 4.2. Pour cela, nous utilisons le même protocole expérimental : nous faisons varier le nombre d'exemples de 5 à 50 et le paramètre de voisinage de 0 à 3.

Le choix des contraintes que nous avons ajoutées aux champs aléatoires conditionnels pour ces deux corpus a été guidé par leur DTD de sortie. On considère l'exemple d'une sous-partie de la DTD de sortie du corpus "Courses" représentée sur la figure 5.3. Dans cette DTD, la première règle donne par exemple lieu à la contrainte suivante :

$$\forall c \in \mathcal{C}_3,\ \forall (y_1, y_2) \in \mathcal{Y}^2 \setminus \{(\texttt{startime}, \texttt{endtime})\},\ (\texttt{time}, y_1, y_2) \notin \mathcal{S}_c$$

De la même façon, à partir de la quatrième règle, on génère la contrainte :

$$\forall c \in \mathcal{C}_3,\ \forall (y_1, y_2) \in \mathcal{Y}^2 \setminus \{(\texttt{building}, \texttt{room})\},\ (\texttt{place}, y_1, y_2) \notin \mathcal{S}_c$$

Exemples	Voisinage	2-CRFs		3-CRFs	
		F1	Gain	F1	Gain
5	0	88.5	+27.41%	**91.89**	+2.1%
	1	91.7	+6.65%	**91.88**	+4.33%
	2	88.27	+1.87%	**90.92**	+7.27%
	3	**92.39**	+3.22%	88.85	+2.06%
10	0	83.72	-3.46%	**92.58**	+0.77%
	1	87.64	-3.78%	**96.58**	+5.49%
	2	89.41	+0.91%	**96.72**	+2.34%
	3	89.80	+1.81%	**96.69**	+4.12%
20	0	79.07	+0.38%	**98.07**	+0.15%
	1	92.65	+0.38%	**96.48**	-0.71%
	2	93.33	-2.66%	**97.69**	-0.03%
	3	94.58	-0.14%	**99.92**	-0.05%
50	0	88.53	-0.55%	**97.74**	-0.83%
	1	96.24	-0.01%	**97.74**	-1%
	2	96.63	-0.21%	**98.74**	-0.52%
	3	96.68	-0.22%	**99.93**	-0.02%

TAB. 5.1 – Résultats avec contraintes pour la tâche d'annotation du corpus "Courses".

Enfin, la règle `<!ELEMENT starttime (#PCDATA)>` signifie que le fils d'un nœud annoté par `startime` sera annoté par $\perp$. De plus, ce fils doit être une feuille. On peut alors générer une contrainte du type :

$$\forall\{n, ni\} \in \mathcal{C}_2,\ feuille(ni) \Rightarrow \forall y \in \mathcal{Y} \setminus \{\perp\},\ (\texttt{starttime}, y) \notin \mathcal{S}_{\{n,ni\}}$$

où $feuille(ni)$ est un prédicat testant que le nœud ni est une feuille. Avec les contraintes générées de cette manière, pour le corpus "Courses", le nombre de combinaisons de labels possibles pour les cliques triangulaires a été réduit de $14^3 = 2744$ à 140. Dans le cas du corpus "Movie" dont l'alphabet des labels est plus grand, le nombre de combinaisons de labels possibles pour les cliques triangulaires a été réduit de $37^3 = 50563$ à 204.

Résultats

Le tableau 5.1 présente les résultats obtenus par les 2-CRFs et les 3-CRFs avec contraintes sur le corpus "Courses". Les résultats présentés correspondent une fois de plus à la macro moyenne des F_1-mesures de chaque label. Nous présentons en plus ici, pour chaque résultat, le gain de performance relatif par rapport aux résultats obtenus sans utiliser les contraintes.

D'un point de vue général, on observe que l'ajout de contraintes aux 2-CRFs et aux 3-CRFs n'occasionnent presqu'aucune diminution des résultats. En effet, dans la moitié des cas, l'ajout de contraintes a permis des gains de performances allant jusque +27.41%, tandis que lorsque les contraintes font baisser les performances, la baisse ne dépasse pas −3.78%, et seulement −1% dans le cas des 3-CRFs. De plus, on remarque aussi que l'ajout de contraintes ne change pas les différences de performances entre les deux modèles de

	2-CRFs		3-CRFs	
	App.	Ann.	App.	Ann.
Temps Moyen sans contraintes	2.75	0.13	15.96	0.24
Temps Moyen avec contraintes	2	0.11	14.25	0.23
Différence	-27.33%	-11.43%	-10.68%	-6.70%

TAB. 5.2 – Temps de calculs moyens (en secondes) pour la tâche d'annotation du corpus "Courses" avec et sans contraintes.

champs aléatoires conditionnels évalués, les 3-CRFs restant plus performants que les 2-CRFs.

En observant ces résultats plus en détails, on remarque que les baisses de performances occasionnées par les contraintes se situent essentiellement lorsque l'échantillon d'apprentissage comporte 20 exemples ou plus, tandis que lorsque moins d'exemples sont utilisés en apprentissage, les contraintes ont tendance à significativement améliorer les résultats. Une explication à ce comportement est que les contraintes, en éliminant des configurations d'annotation possibles, guident le champ aléatoire conditionnel dans ses choix d'annotation. Ainsi, lorsque l'échantillon d'apprentissage ne comporte pas suffisamment d'exemples pour apprendre un modèle correct, l'utilisation de contraintes permet d'éviter les erreurs qu'aurait commises le champ aléatoire conditionnel.

Le tableau 5.2 présente les temps de calcul moyens en secondes des 2-CRFs et 3-CRFs avec contraintes sur le corpus "Courses". Tous les temps de calcul sont rapportés au nombre de documents : le temps d'apprentissage est divisé par le nombre d'exemples, et le temps d'annotation est le temps moyen par document. Les temps affichés correspondent à une moyenne sur toutes les expériences dont les résultats sont présentés dans le tableau 5.1.

Les temps de calcul observés avec l'utilisation de contraintes sont bien inférieurs aux temps observés sur le même corpus avec les champs aléatoires conditionnels sans contrainte. Ce gain de temps atteint même les 22.73% dans le cas de l'apprentissage des 2-CRFs. Toutefois, on peut noter que le gain de temps est très sensiblement plus faible que le gain théorique annoncé. Ceci est dû à l'implantation. En effet, d'un côté, l'implantation des champs aléatoires conditionnels a été optimisée au maximum et permet d'éviter d'effectuer des calculs inutiles. De l'autre, l'ajout de contraintes aux champs aléatoires conditionnels impose d'effectuer des tests supplémentaires lors des calculs, augmentant le temps de calcul. Ces deux facteurs font que le gain observé lors de l'utilisation de contraintes est plus faible que prévu.

Nous présentons maintenant les résultats pour la deuxième série d'expériences de renommage de nœuds sur le corpus "Movie". Le tableau 5.3 présente les résultats des 2-CRFs et 3-CRFs avec contraintes pour cette nouvelle série d'expériences. Contrairement aux expériences sur le corpus "Courses", les champs aléatoires conditionnels obtenaient déjà d'excellents résultats sur le corpus "Movie" sans l'utilisation de contraintes. On remarque toutefois que, même si le gain de performances est moins significatif, les contraintes permettent aux 2-CRFs et aux 3-CRFs d'améliorer encore ces résultats. En effet, dans quasiment toutes les expériences effectuées, on observe un gain allant jusqu'à +4%, les quelques baisses de performances que l'on constate ne dépassant pas −0.80%. Si l'on observe plus

Exemples	Voisinage	2-CRFs		3-CRFs	
		F1	Gain	F1	Gain
5	0	**99.97**	+2.60%	99.57	+0.31%
	1	**99.64**	+0.42%	99.47	+0.14%
	2	**99.43**	+1.09%	98.35	-0.80%
	3	98.47	-0.35%	**99.81**	+1.16%
10	0	98.78	+4.07%	**99.07**	+0.42%
	1	**99.14**	+0.54%	98.85	+0.03%
	2	98.99	+0.70%	**99.07**	+0.42%
	3	98.71	+0.16%	**99.26**	+0.84%
20	0	**99.32**	+3.77%	98.73	+3.73%
	1	98.96	+2.52%	**99.08**	+2.63%
	2	**99.37**	+1.20%	99.12	+0.43%
	3	**99.53**	+1.93%	98.84	+0.14%
50	0	**99.62**	+4.16%	99.60	-0.02%
	1	99.53	+0.72%	**99.74**	+0.19%
	2	99.71	+0.11%	**100**	+0.35%
	3	**99.39**	-0.22%	99.21	-0.51%

TAB. 5.3 – Résultats avec contraintes pour la tâche d'annotation du corpus "Courses".

	2-CRFs		3-CRFs	
	App.	Ann.	App.	Ann.
Temps Moyen sans contraintes	57.93	1.61	383.35	1.98
Temps Moyen avec contraintes	15.5	0.68	283.86	0.81
Différence	-73.24%	-57.93%	-25.95%	-58.85%

TAB. 5.4 – Temps de calculs moyens (en secondes) pour la tâche d'annotation du corpus "Movie" avec et sans contraintes.

en détail ces résultats, on remarque que le gain est en général plus important dans les expériences où le paramètre de voisinage est petit (0 ou 1). Ceci n'est pas surprise. En effet, avec un paramètre de voisinage faible, la quantité d'informations sur l'observation est restreinte et peu de connaissances sont donc utilisables. L'ajout de contraintes permet d'intégrer des connaissances du domaine, sur la sortie cette fois-ci, et donc de contrebalancer le manque d'informations sur l'entrée.

Nous nous intéressons maintenant aux temps de calcul des 2-CRFs et 3-CRFs avec contraintes sur le corpus "Movie", présentés dans le tableau 5.4. On remarque que sur ce corpus encore, les contraintes ont permis des temps de calcul moyens nettement plus courts. En effet, on observe des gains de temps allant jusqu'à -73.24% en apprentissage pour les 2-CRFs. De plus, le temps d'annotation moyen d'un document, avec les 2-CRFs ou les 3-CRFs, a été plus que réduit de moitié. La taille de l'alphabet des labels sur le corpus "Movie" explique que les contraintes ont apporté ici un gain de temps de calcul plus important. En effet, avec 37 labels, contre 14 pour le corpus "Courses", la DTD de sortie a permis d'intégrer un plus grand nombre de contraintes et donc de réduire plus

fortement la complexité de l'estimation des paramètres ainsi que de l'annotation.

5.1.6 Bilan sur les contraintes

Les deux séries d'expériences sur les corpus "Courses" et "Movie" montrent donc bien l'intérêt des contraintes. En effet, non seulement celles-ci achèvent leur but premier qui était d'améliorer la complexité en termes de temps de calcul à la fois lors de l'apprentissage et de l'annotation. Mais ces contraintes ont aussi permis, dans la plupart des cas, d'améliorer, parfois significativement, les résultats obtenus sans contraintes, particulièrement dans les cas où on dispose de peu d'exemples en apprentissage ou lorsque peu d'informations sur l'observation sont à disposition.

Toutefois, on remarque d'une part que les gains de complexité en temps, s'ils atteignent jusqu'à 73% en apprentissage dans le cas des 2-CRFs, ne permettent pas encore d'utiliser les champs aléatoires conditionnels dans des applications *online*. De plus, alors que la complexité en temps est améliorée, la complexité en espace mémoire, pour sa part, reste inchangée.

Nous présentons maintenant une deuxième méthode permettant d'améliorer la complexité à la fois en termes de temps de calcul, et en termes d'espace mémoire. Tandis que les contraintes ne changent pas le modèle considéré et modélisent directement la probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$ de l'annotation $\mathbf{y}$ sachant l'observation $\mathbf{x}$, la nouvelle méthode consiste à approximer cette probabilité conditionnelle à l'aide de plusieurs autres probabilités conditionnelles plus simples, elles aussi modélisées par des champs aléatoires conditionnels.

5.2 Composition de CRFs

Nous abordons maintenant la deuxième méthode d'amélioration de la complexité des champs aléatoires conditionnels que nous proposons. Celle-ci s'appuie sur le fait que, dans de nombreux cas, une structuration naturelle de l'alphabet des labels existe. Cette structuration peut être soit une simple partition de l'alphabet, c'est-à-dire une décomposition en plusieurs sous-parties que l'on peut considérer comme étant indépendantes les unes des autres, soit une véritable structure hiérarchique sous forme d'arbre. C'est le cas par exemple des labels inclus dans des ontologies, ou, dans le domaine qui nous intéresse ici, lors de l'annotation d'arbre avec des labels correspondant à une DTD. Tenir compte de cette structuration de l'alphabet des labels permet alors d'approximer un champ aléatoire conditionnel défini sur l'intégralité de cet alphabet par plusieurs champs aléatoires conditionnels définis sur des sous-ensembles de $\mathcal{Y}$. Nous proposons, dans ce sens, trois méthodes de composition de champs aléatoires conditionnels différentes. D'une part, les deux premières méthodes s'appuient sur l'existence d'une partition de l'alphabet des labels. Dans ce cas, une première solution consiste à considérer que chaque sous-partie de l'alphabet des labels, et par conséquent chaque champ aléatoire conditionnel défini sur cette sous-partie, est indépendant, tandis que la seconde considère que chaque sous-partie de l'alphabet des labels peut dépendre des sous-parties la précédant. D'autre part, la troisième méthode combine les deux premières et s'appuie sur l'existence d'une hiérarchie sur

```
<!ELEMENT movie (presentation,tagLine,plots,rating,cast,technics,stories)>
<!ELEMENT presentation (fullTitle,usualTitle,year,directors,wc)>
<!ELEMENT directors (name+)>
<!ELEMENT wc (name+)>
<!ELEMENT plots (plot+)>
<!ELEMENT plot (author?,text)>
<!ELEMENT rating (p?,score,vote?)>
<!ELEMENT p (img+)>
<!ELEMENT cast (c+)>
<!ELEMENT c (actor,caracter)>
<!ELEMENT technics (runtime,country,language,colors,sounds,certificates)>
<!ELEMENT colors (color+)>
<!ELEMENT sounds (sound+)>
<!ELEMENT certificates (certificate+)>
<!ELEMENT stories (trivias,goofs)>
<!ELEMENT trivias (trivia+)>
<!ELEMENT goofs (goof+)>
```

FIG. 5.4 – DTD de la tâche d'annotation du corpus "Movie".

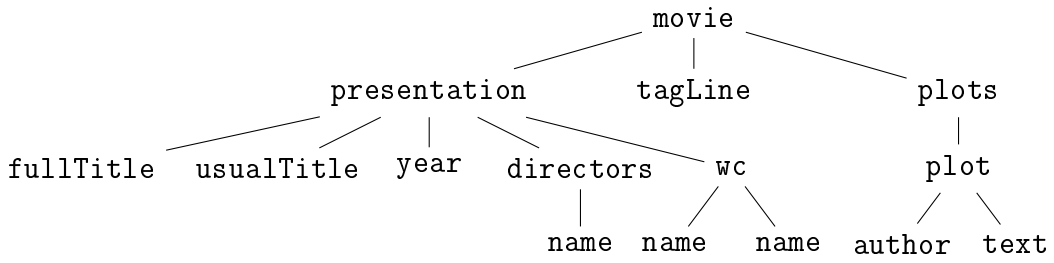

FIG. 5.5 – Exemple partiel d'arbre annotation pour le corpus "Movie". Les nœuds textes ne sont pas représentés.

l'alphabet des labels.

Nous présentons maintenant ces trois méthodes de composition de champs aléatoires conditionnels. Ces méthodes supposent l'existence d'une procédure de génération automatique des fonctions de caractéristiques que nous notons Gen. Elle prend en paramètres un ensemble de couples (observation,annotation) et retourne un ensemble de fonctions de caractéristiques. Son fonctionnement est décrit en détail dans l'annexe A. Lors de la présentation des trois méthodes de composition, nous considérerons l'exemple de la tâche d'annotation des nœuds du corpus "Movie" présenté dans la section 4.3.1, qui comporte un alphabet de 37 labels correspondant aux 37 éléments de la DTD présentée en figure 5.4 (les définitions d'éléments ne contenant que du texte ont été enlevées) et dont un exemple partiel d'arbre est représenté sur la figure 5.5. Nous présentons ensuite une série d'expériences sur des tâches d'annotation d'arbres permettant de comparer ces méthodes et mettant en évidence à la fois le gain de complexité qu'elles occasionnent et la stabilité de leurs performances en les comparant aux champs aléatoires conditionnels traditionnels.

5.2.1 Méthodes de composition avec une partition de l'alphabet des labels

Nous présentons donc, dans un premier temps, les méthodes de composition de champs aléatoires conditionnels s'appuyant sur la connaissance d'une partition de l'alphabet des labels $\mathcal{Y}$ de taille k : $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$.

Composition parallèle

La première méthode de composition que nous proposons est la **composition parallèle**. Cette méthode considère que l'on peut trouver une partition de l'alphabet des labels $\mathcal{Y}$. Les annotations pour chacune des k parties de l'alphabet des labels sont considérées indépendantes. Sur chaque partie de $\mathcal{Y}$, un champ aléatoire conditionnel est défini. Celui-ci utilise les labels de la partie correspondante ainsi qu'un label supplémentaire que nous notons $\perp$. Lorsqu'un nœud est annoté par le label $\perp$, celui-ci doit être annoté par un champ aléatoire conditionnel correspondant à une autre sous-partie de l'alphabet des labels.

Soient $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ une partition de $\mathcal{Y}$, $\perp$ un label spécial et $\mathcal{C} = (F, \Lambda)$ un champ aléatoire conditionnel dont l'alphabet des labels est $\mathcal{Y}$ où F et Λ sont respectivement un ensemble de fonctions de caractéristiques et les poids qui leurs sont associés. Par composition parallèle, le champ aléatoire conditionnel $\mathcal{C}$ peut être approximé par un ensemble de k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$ dont l'alphabet des labels est $\mathcal{Y}_i \cup \{\perp\}$. Pour chaque $\mathcal{C}_i$, les observations ne sont pas modifiées et celles-ci sont définies sur l'alphabet des symboles $\mathcal{X}$.

Sur l'exemple de la tâche d'annotation dont la DTD de sortie est représentée en figure 5.4, une partition possible de l'alphabet des labels $\mathcal{Y}$ consiste à créer une sous-partie de l'alphabet des labels pour chaque sous-arbre partant d'un fils de la racine, ainsi qu'une sous-partie pour la racine elle-même. On obtient ainsi la partition suivante :

$$\mathcal{Y}_1 = \{\texttt{movie}\}$$
$$\mathcal{Y}_2 = \{\texttt{presentation}, \texttt{fullTitle}, \texttt{usualTitle}, \texttt{year}, \texttt{directors}, \texttt{wc}, \texttt{name}\}$$
$$\mathcal{Y}_3 = \{\texttt{tagLine}\}$$
$$\mathcal{Y}_4 = \{\texttt{plots}, \texttt{plot}, \texttt{author}, \texttt{text}\}$$
$$\ldots$$

Toutes les sous-parties de la partition ne sont pas représentées ici. Cette partition est constituée, au total, de 8 sous-parties dont la taille varie de 1 à 10 labels. Avec cette partition, on obtient l'arbre annotation représenté sur la figure 5.5 en appliquant plusieurs CRFs en parallèle. Le CRF $\mathcal{C}_1$ annote la racine par le label `movie`, et tous les autres nœuds par $\perp$. Le résultat de l'annotation par le CRF $\mathcal{C}_2$ est représenté sur la figure 5.6. Conformément à la façon dont a été construite la partition, seul le premier sous-arbre est annoté par des labels de $\mathcal{Y}_2$, tous les autres nœuds étant annotés par $\perp$.

Nous considérons maintenant l'algorithme 2 d'apprentissage pour la composition parallèle. Celui-ci prend en paramètres un échantillon d'apprentissage $S = \{(\mathbf{x}^j, \mathbf{y}^j)\}_{j=1}^{j=m}$, la partition de l'alphabet des label ainsi qu'une procédure de génération de fonctions de caractéristiques Gen. À chaque étape i, l'algorithme commence par générer l'ensemble

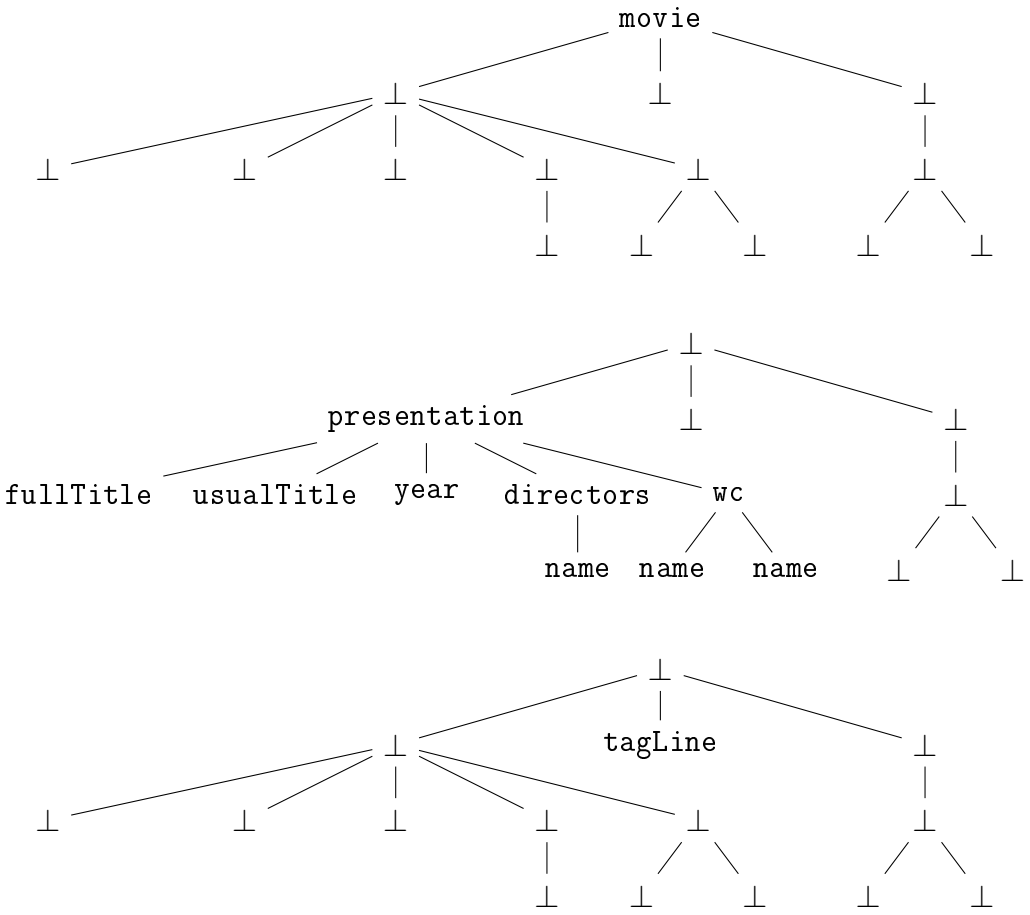

FIG. 5.6 – Annotations de l'arbre de la figure 5.5 aux étapes 1, 2 et 3 (de haut en bas) de la composition parallèle. Les nœuds texte ne sont pas représentés.

d'apprentissage S_i correspondant. Pour cela, dans toutes les annotations $\mathbf{y}^j$ de l'ensemble d'apprentissage S, la fonction Cpl_i remplace tous les labels l tels que $l \notin \mathcal{Y}_i$ par le label $\perp$. On définit ainsi Cpl_i de la façon suivante :

$$\mathrm{Cpl}_i(y) = \begin{cases} y \text{ si } y \in \mathcal{Y}_i \\ \perp \text{ sinon} \end{cases}$$

On étend ensuite la fonction Cpl aux vecteurs, et on obtient l'ensemble d'apprentissage à l'étape i : $S_i = \{(\mathbf{x}, \mathrm{Cpl}_i(\mathbf{y}))|(\mathbf{x}, \mathbf{y}) \in S\}$. L'ensemble des fonctions de caractéristiques F_i est ensuite généré à partir de S_i à l'aide de la procédure de génération Gen. L'algorithme d'apprentissage traditionnel des champs aléatoires conditionnels peut ensuite être appliqué. Chaque étape i étant indépendante des autres, les différentes étapes de cette méthode de composition peuvent être effectuées en parallèle, améliorant ainsi nettement le temps de calcul.

Algorithme 2 Apprentissage pour la composition parallèle.

Entrée: Un échantillon d'apprentissage S de couple $\{(\mathbf{x}^j, \mathbf{y}^j)\}_{j=1}^{j=m}$, la partition de l'alphabet des labels $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$, une procédure de génération de fonctions de caractéristiques Gen.

1: **pour chaque** $i \in \{1, \ldots, k\}$ **do**
2: S_i : dans tous les $\mathbf{y}^j$, remplacer tous les labels $l \notin \mathcal{Y}_i$ par $\perp$
3: $F_i = \mathrm{Gen}(S_i)$ *# générer l'ensemble des fonctions de caractéristiques*
4: $\Lambda_i = \mathrm{TrainCRF}(S_i, F_i)$ *# Entraîner le champ aléatoire conditionnel*
5: **fin pour**

Sortie: k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$

L'algorithme d'annotation 3 pour la composition parallèle est très similaire. Pour annoter une observation $\mathbf{x}$ donnée, les k champs aléatoires conditionnels obtenus précédemment sont appliqués simultanément. Les résultats sont ensuite recombinés dans une seule annotation au moyen de la fonction Combine. Cette fonction regarde, pour chaque variable aléatoire Y_n, le vecteur v_n des annotations qui lui ont été affectées par les k champs aléatoires conditionnels. Dans le cas parfait, ce vecteur possède $k-1$ fois la valeur $\perp$ et un label $l \in \mathcal{Y}$: ce label l est alors choisi. Toutefois, il peut arriver que plusieurs champs aléatoires conditionnels entrent en conflit, c'est-à-dire que deux labels différents de $\perp$ soient présents dans le vecteur d'annotation. Dans ce cas, on choisit le label qui a la meilleure probabilité marginale, c'est-à-dire le label l_i tel que :

$$l_i = \underset{1 \leq i \leq k, l_i \neq \perp}{\arg\max}\ p_{\mathcal{C}_i}(Y_n = l_i|\mathbf{x})$$

où $p_{\mathcal{C}_i}(Y_n = l_i|\mathbf{x})$ est la probabilité marginale du label l_i affecté à la variable aléatoire Y_n dans le champ aléatoire conditionnel $\mathcal{C}_i$.

Cette combinaison permet donc de réduire grandement la complexité de l'annotation et de l'apprentissage. En effet, le facteur $|\mathcal{Y}|^K$ présent dans la complexité de ces deux algorithmes, où K est la taille des cliques maximales du graphe d'indépendances, est

Algorithme 3 Annotation pour la composition parallèle.

Entrée: une entrée $\mathbf{x}$, k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$, la partition de l'alphabet des labels $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$
pour chaque $i \in \{1, \ldots, k\}$ **do**
  $\mathbf{y}_i = \text{LabelCRF}(\mathbf{x}, \mathcal{C}_i)$
fin pour
$\hat{\mathbf{y}} = \text{Combine}(\mathbf{y}_1, \ldots, \mathbf{y}_k)$
Sortie: $\hat{\mathbf{y}}$

alors réduit à $(\max_{1 \leq i \leq k})|\mathcal{Y}_i|^K$. Si on suppose que les k parties $\mathcal{Y}_i$ ont la même cardinalité, ce facteur de complexité est alors réduit à $(\frac{|\mathcal{Y}|}{k})^K$.

Sur l'exemple de la tâche d'annotation du corpus "Movie", la partition de l'alphabet des labels que nous avons définie précédemment se compose de 8 sous-parties, dont la plus grande (celle correspondant au sous-arbre partant du label `technics`) est de taille 10. La composition parallèle a donc permis de réduire le facteur de complexité $|\mathcal{Y}|^K$ de $37^3 = 50653$ à $10^3 = 1000$ (50 fois moins).

On peut toutefois remarquer sur cet exemple une des limitations de la composition parallèle. En effet, en observant la DTD de la figure 5.4, on remarque que les labels `presentation` et `tagLine` sont deux frères consécutifs et sont donc fortement dépendants. Avec les champs aléatoires conditionnels sans composition, ces deux labels se retrouvent dans une même clique et le choix de l'un peut donc influer sur l'autre. Toutefois, dans le cas de la composition parallèle, la dépendance entre ces deux labels est brisée, ceux-ci appartenant à deux sous-parties différentes. Afin de remédier à ce problème, nous présentons maintenant un seconde méthode de composition basée elle aussi sur une partition de l'alphabet des labels $\mathcal{Y}$.

Composition séquentielle

Cette deuxième méthode de composition de champs aléatoires conditionnels que nous proposons est la méthode de **composition séquentielle**. Celle-ci repose sur le même principe que la composition parallèle et utilise une partition de l'alphabet des labels $\mathcal{Y}$. Toutefois, on suppose cette fois qu'il existe un ordre total sur cette partition. Le choix de cet ordre peut provenir de différentes sources. La connaissance a priori de la difficulté de l'annotation de chacune des sous-parties de $\mathcal{Y}$ peut être un premier facteur de choix de l'ordre (du plus simple au plus difficile). Un autre critère de choix de l'ordre peut être l'existence d'une DTD sur les labels : une branche de la DTD donne un ordre sur les labels, l'ordre des sous-arbres de la racine est un autre choix possible d'ordre sur les sous-parties de $\mathcal{Y}$.

Le principe de la composition séquentielle est d'appliquer les différents champs aléatoires conditionnels les uns après les autres selon l'ordre établi sur la partition de l'alphabet des labels. Soit $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ une partition ordonnée de $\mathcal{Y}$. Un champ aléatoire conditionnel $\mathcal{C}$ dont l'alphabet des labels est $\mathcal{Y}$ peut être approximé par k champs aléatoires conditionnels $\mathcal{C}_i$ dont l'alphabet des labels est $\mathcal{Y}_i \cup \{\perp\}$. L'intérêt de cette nouvelle méthode de composition par rapport à la composition parallèle est qu'elle permet de réintroduire des dépendances sur des labels n'appartenant pas à la même sous-partie dans la partition de

l'alphabet des labels $\mathcal{Y}$. En effet, comme les différents champs aléatoires sont utilisés en séquence, il devient alors possible de supposer que l'annotation effectuée à l'étape i peut être aidée par les annotations choisies aux étapes précédentes j telles que $1 \leq j < i$. Pour ce faire, les résultats des annotations effectuées aux étapes précédentes sont intégrées à l'observation $\mathbf{x}$. Ceci permet alors de simuler des dépendances dirigées dans les champs aléatoires conditionnels. De plus, à la différence des dépendances classiques des champs aléatoires conditionnels, ces dépendances simulées peuvent être des dépendances longue distance, c'est-à-dire entre des variables aléatoires n'appartenant pas nécessairement à la même clique.

Nous illustrons maintenant cette méthode sur l'exemple de la tâche d'annotation du corpus "Movie". La même partition que celle employée pour la composition parallèle est considérée. Nous définissons de plus un ordre total sur les sous-parties provenant de l'ordre des fils de la racine (*cf.* la première ligne de la DTD de la figure 5.4). On a ainsi $\mathcal{Y}_1 < \mathcal{Y}_2 < \cdots < \mathcal{Y}_8$. Ainsi, avec la méthode de composition séquentielle, les arbres d'entrée à chaque étape i sont alors définis sur les alphabets d'étiquette suivants :

$$\begin{aligned}
\mathcal{X}_1 &= \mathcal{X} \times \emptyset \\
\mathcal{X}_2 &= \mathcal{X} \times \{\texttt{movie}, \bot\} \\
\mathcal{X}_3 &= \mathcal{X} \times \{\texttt{movie}, \texttt{presentation}, \texttt{fullTitle}, \texttt{usualTitle}, \texttt{year}, \\
&\qquad \texttt{directors}, \texttt{wc}, \texttt{name}, \bot\} \\
&\ldots
\end{aligned}$$

À l'étape 2, l'alphabet d'entrée est $\mathcal{X}_2 = \mathcal{X} \times \{\texttt{movie}, \bot\}$ car le CRF $\mathcal{C}_1$, dont l'alphabet des labels est $\mathcal{Y}_1 = \{\texttt{movie}, \bot\}$ a déjà été appliqué et son résultat est intégré à l'observation. Ainsi, si l'on considère la racine, on a à la première étape $x_\epsilon = (\texttt{NODE}, \bot)$, et à toutes les étapes suivantes $x_\epsilon = (\texttt{NODE}, \texttt{movie})$. De la même façon, pour le premier fils de la racine, qui doit être annoté par `presentation`, on a $x_1 = (\texttt{NODE}, \bot)$ aux deux premières étapes, puis $x_1 = (\texttt{NODE}, \texttt{presentation})$ aux étapes suivantes. En intégrant progressivement à l'observation les annotations effectuées par les CRFs à chaque étape, la composition séquentielle permet d'utiliser ces annotations aux étapes suivantes. Grâce à cela, certaines limitations de la composition parallèle sont ainsi corrigées. En effet, des dépendances dirigées sont ajoutées grâce à l'ordre sur les sous-parties. Par exemple, le choix du label `technics`, qui appartient à la sous-partie $\mathcal{Y}_7$, peut être conditionné par les labels des sous-parties $\mathcal{Y}_1$ à $\mathcal{Y}_6$, comme par exemple `presentation` ou `actor`. Les dépendances ainsi simulées possèdent l'intérêt de ne pas être limitées au simple cadre de la clique. La composition séquentielle fait donc plus qu'approximer un champ aléatoire conditionnel classique, car elle ajoute aussi des dépendances. Toutefois, certaines dépendances existant dans le champ aléatoire conditionnel d'origine sont perdues. Par exemple, si le choix du label `tagLine` peut toujours dépendre du label `presentation`, l'inverse n'est plus vrai car ce dernier appartient à une sous-partie de l'alphabet des labels qui se trouve avant celle dont fait partie `tagLine`.

Pour cette méthode de composition, on définit tout d'abord un algorithme d'apprentissage (algorithme 4). Celui-ci prend en paramètres un échantillon d'apprentissage S, la partition ordonnée de l'alphabet des labels, ainsi que la procédure de génération des fonctions de caractéristiques Gen. À chaque étape i, l'algorithme commence par générer le

corpus d'apprentissage S_i. Pour cela, les observations sont définies sur l'union de l'alphabet des symboles $\mathcal{X}$ et des sous-parties de l'alphabet $\mathcal{Y}_j$ telles que $1 \leq j < i$. L'alphabet des symboles à l'étape i se note donc $\mathcal{X}_i = \mathcal{X} \times (\bigcup_{1 \leq j < i} \mathcal{Y}_j \cup \{\perp\})$. L'alphabet des labels à l'étape i est $\mathcal{Y}_i \cup \perp$. Plus formellement, si l'on considère un couple défini sur $\mathcal{X} \times \mathcal{Y}$, la fonction π_i génère l'observation correspondant à ce couple à l'étape i de l'algorithme :

$$\pi_i(x, y) = \begin{cases} (x, y) \text{ si } y \in \mathcal{Y}_j \text{ tel que } 1 \leq < i \\ (x, \perp) \text{ sinon} \end{cases}$$

En étendant cette fonction aux vecteurs, l'ensemble d'apprentissage à l'étape i est défini de la façon suivante : $S_i = \{(\pi_i(\mathbf{x}, \mathbf{y}), \mathrm{Cpl}_i(\mathbf{y})) \mid (\mathbf{x}, \mathbf{y}) \in S\}$. À partir de cet ensemble S_i, l'algorithme d'apprentissage construit ensuite l'ensemble des fonctions de caractéristiques F_i. Ces fonctions sont définies sur des observations prenant leurs valeurs dans $\mathcal{X}_i$. Par conséquent, elles considèrent non seulement l'observation $\mathbf{x}$ mais aussi l'ensemble des labels affectés aux étapes précédentes. De plus, afin d'éviter les conflits obtenus lors de l'annotation avec la méthode de composition parallèle, nous ajoutons à chaque champ aléatoire conditionnel $\mathcal{C}_i$ des contraintes locales de la forme :

$$\forall n \in \mathcal{C}_1, \ \Big(x_n = (x, y) \text{ tel que } y \neq \perp\Big) \ \Rightarrow \forall l \in \mathcal{Y}_i, \ l \notin \mathcal{S}_n$$

Cette contrainte impose que seules les variables aléatoires dont la deuxième composante de l'observation est $\perp$ peuvent être annotées par un label appartenant à $\mathcal{Y}_i$. Dans le cas contraire, cette variable aléatoire est obligatoirement annotée par $\perp$.

Algorithme 4 Apprentissage pour la composition séquentielle.

Entrée: un échantillon d'apprentissage S composé de couples $\{(\mathbf{x}^j, \mathbf{y}^j)\}_{j=1}^{j=m}$, une fonction de génération de fonctions de caractéristiques Gen.

1: **pour** i de 1 à k **do**
2: $S_i = \{(\pi_i(\mathbf{x}, \mathbf{y}), \mathrm{Cpl}_i(\mathbf{y})) \mid (\mathbf{x}, \mathbf{y}) \in S\}$
3: $F_i = \mathrm{Gen}(S_i)$ `# générer l'ensemble de fonctions de caractéristiques`
4: Ajout des contraintes locales
5: $\Lambda_i = \mathrm{TrainCRF}(S_i, F_i)$ `# estimation des paramètres`
6: **fin pour**

Sortie: k CRFs $\mathcal{C}_i = (F_i, \Lambda_i)$

L'algorithme 5 d'annotation pour la composition séquentielle prend quant à lui en paramètres une observation $\mathbf{x}$ ainsi que k champs aléatoires conditionnels $\mathcal{C}_i$. Ces k champs aléatoires conditionnels sont appliqués les uns à la suite des autres sur l'observation afin d'obtenir l'annotation finale. Pour cela, $\mathcal{C}_1$ est dans un premier temps appliqué sur $\mathbf{x}$ et produit la première annotation. Ensuite, à chaque étape $i > 1$, l'observation $\mathbf{x}_i$ correspond au couple $((\mathbf{x}, \mathbf{y}_{i-1}))$ défini sur $\mathcal{X}_i = \mathcal{X} \times (\bigcup_{1 \leq j < i} \mathcal{Y}_j \cup \{\perp\})$. $\mathbf{y}_{i-1}$ correspond donc à la combinaison des annotations effectuées par les champs aléatoires conditionnels $\mathcal{C}_j$ pour $1 \leq j < i$. Les contraintes introduites lors de l'apprentissage empêchent toute possibilité de conflit lors de la combinaison des annotations. La combinaison consiste ici simplement à remplacer le label $\perp$ par le label affecté à l'étape i, si celui-ci est différent de $\perp$. Ainsi,

à chaque étape i, l'annotation $\mathbf{y}_i$ produite correspond à l'annotation avec les i premières sous-parties de la partition, contrairement à la composition parallèle où chaque annotation à l'étape i correspond uniquement aux labels de la sous-partie $\mathcal{Y}_i$. En guise d'exemple, la figure 5.2.1.0 représente les annotations produites aux étapes 1, 2 et 3 dans le cadre de la tâche d'annotation du corpus "Movie". Le résultat final $\mathbf{y}_k$ de l'algorithme 5 correspond alors à la combinaison des annotations effectuées par les k champs aléatoires conditionnels.

Algorithme 5 Annotation pour la composition séquentielle.

Entrée: une observation $\mathbf{x}$, la partition de l'alphabet des labels $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ et k CRFs avec contraintes $\mathcal{C}_i = (F_i, \Lambda_i)$.

1: $\mathbf{y}_1 = \text{LabelCRF}(\mathbf{x}, \mathcal{C}_1)$
2: **pour** i de 2 à k **do**
3: $\quad \mathbf{x}_i = (\mathbf{x}, \mathbf{y}_{i-1})$
4: $\quad res_i = \text{LabelCRF}(\mathbf{x}_i, \mathcal{C}_i)$
5: $\quad \mathbf{y}_i = combine(\mathbf{y}_{i-1}, res_i)$
6: **fin pour**

Sortie: $\mathbf{y}_k$

Avec cette méthode de composition séquentielle, la complexité de l'algorithme d'apprentissage reste la même que pour la composition parallèle. En effet, les k corpus d'apprentissage peuvent être générés indépendamment et les k étapes d'apprentissage peuvent être lancées en parallèle. En supposant les k sous-parties de l'alphabet des labels $\mathcal{Y}$ de même taille, le facteur de complexité $|\mathcal{Y}|^K$ de l'algorithme d'apprentissage devient $(\frac{|\mathcal{Y}|}{k}^K)$. Il n'en est par contre pas de même pour l'algorithme d'annotation. En effet, chaque étape i de cet algorithme nécessite le résultat de l'étape précédente $i-1$. Les différentes annotations doivent donc être lancées successivement. Toujours en supposant les k sous-parties de l'alphabet des labels de même taille, on réduit, pour l'algorithme d'annotation, le facteur de complexité à $k \times (\frac{|\mathcal{Y}|}{k}^K)$.

La composition séquentielle comporte toutefois un dernier défaut majeur. En effet, si l'apport des nouvelles dépendances est important, il est aussi fortement dépendant du choix de l'ordre défini sur la partition de l'alphabet des labels $\mathcal{Y}$. C'est pourquoi nous proposons une dernière méthode de composition permettant de tirer partie de ces nouvelles dépendances sans avoir à faire de choix "arbitraire" sur l'ordre de la partition.

5.2.2 Composition hiérarchique

La dernière méthode de composition de champs aléatoires conditionnels que nous proposons est la **composition hiérarchique**. Contrairement aux deux premières méthodes de composition proposées, celle-ci ne s'appuie pas sur l'existence d'une partition de l'alphabet. En effet, elle considère l'existence d'une hiérarchie sur les labels et déduit automatiquement de cette hiérarchie une partition. Nous présentons dans un premier temps le principe général de cette méthode avec un exemple, avant de décrire les algorithmes d'apprentissage et d'annotation et d'en déduire leur complexité.

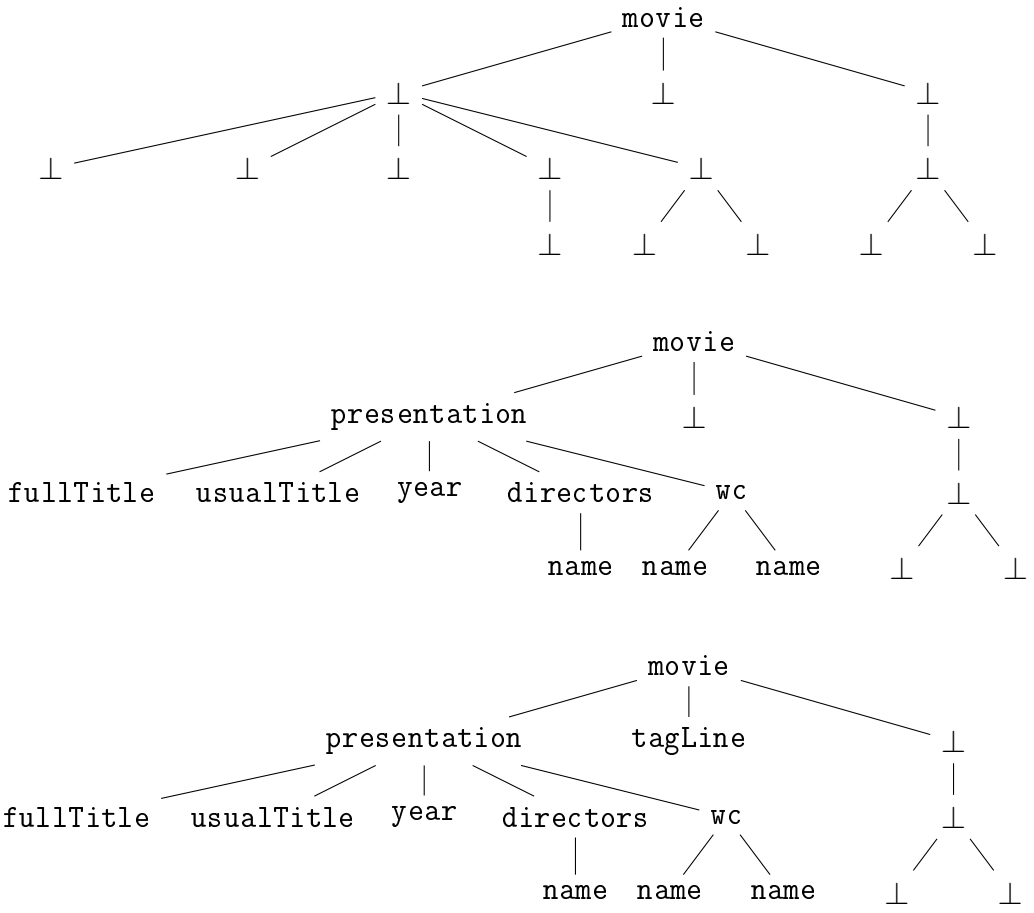

FIG. 5.7 – Annotations de l'arbre de la figure 5.5 aux étapes 1, 2 et 3 (de haut en bas) de la composition séquentielle.

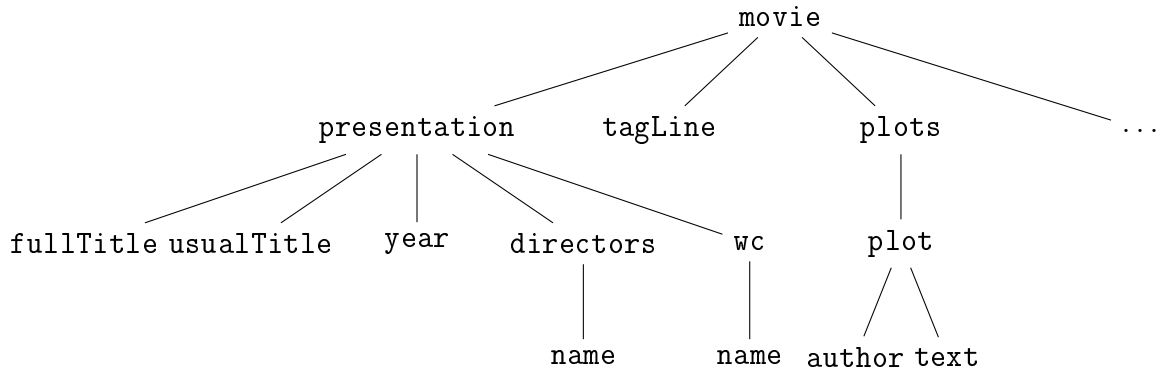

FIG. 5.8 – Représentation arborescente partielle de la DTD de la figure 5.4.

Principe général

D'un point de vue général, le principe de cette méthode est de supposer l'existence d'une hiérarchie sur l'alphabet des labels $\mathcal{Y}$ afin de définir automatiquement à la fois comment séparer l'alphabet des labels en plusieurs sous-parties, mais aussi afin de choisir dans quel ordre utiliser ces sous-parties. Par exemple, dans le cadre de la tâche d'annotation du corpus "Movie", la DTD des arbres XML de sortie constitue en soi une hiérarchie sur les labels de la tâche d'annotation. Cette DTD, puisqu'elle ne contient pas de récursion verticale, peut par exemple être représentée sous la forme d'un arbre comme le montre la figure 5.8.

Une telle hiérarchie permet alors de considérer un découpage de l'alphabet des labels. Dans une première étape, on effectue l'annotation par la racine de cette hiérarchie, en l'occurrence `movie`. Puis, l'annotation par les labels fils de la racine dans la hiérarchie ($\{$`presentation`, `tagLine`, `plots`, ...$\}$) est effectuée. Pour chaque label de cette ensemble est effectuée l'annotation par l'ensemble des fils de ce label dans la hiérarchie. Cette algorithme continue récursivement jusqu'en bas de la hiérarchie. On dispose alors d'une représentation arborescente des alphabets des labels, comme représenté sur la figure 5.9, à utiliser à chaque étape de la composition hiérarchique. Sur cette représentation arborescente des alphabets des labels, on remarque des liens entre les différents alphabets qui nous permettent de choisir l'ordre dans lequel effectuer les tâches d'annotations. En effet, la relation "père-fils" entre les alphabets des labels donne comme indication que l'annotation avec l'alphabet fils doit pouvoir utiliser le résultat de l'annotation avec l'alphabet père. Par exemple, l'annotation avec l'alphabet $\mathcal{Y}_{\texttt{plots}}$ doit être effectuée en ayant connaissance du résultat de l'annotation avec l'alphabet $\mathcal{Y}_{\texttt{movie}}$, principalement pour que les labels `plot` puissent être affectés en sachant quels nœuds ont été annotés par `plots`. À l'inverse, l'observation de la représentation arborescente des alphabets des labels montre que les annotations avec des alphabets frères peuvent être effectuées indépendamment. Par exemple, le résultat de l'annotation avec l'alphabet $\mathcal{Y}_{\texttt{presentation}}$ n'a pas besoin d'être connu pour annoter avec l'alphabet $\mathcal{Y}_{\texttt{plots}}$. Ainsi, ces étapes d'annotation peuvent être effectuées en parallèle.

En combinant ainsi les méthodes de composition parallèle et séquentielle, on obtient des dépendances entre labels plus intuitives. En effet, on a d'une part des dépendances

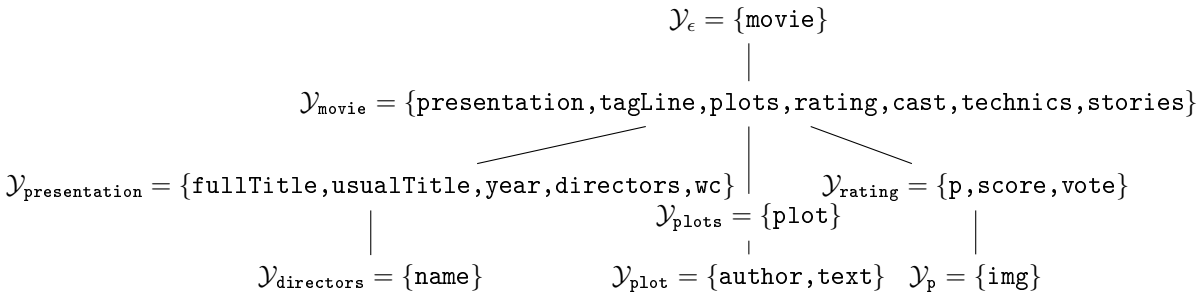

FIG. 5.9 – Représentation arborescente partielle des ensembles de labels provenant de la hiérarchie de labels de la figure 5.8.

locales non-dirigées au niveau des cliques entre les labels qui sont frères dans la hiérarchie, et d'autre part, l'aspect séquentiel permet de simuler des dépendances longue distance dirigées entre un label et tous ses descendants. En utilisant la hiérarchie de labels de la figure 5.8, on simule par exemple une dépendance longue distance entre les labels `plots` et `author`, alors que dans la cas de la composition séquentielle, la présence de cette dépendance dépendait du choix de la partition de l'alphabet des labels. À l'inverse, aucune dépendance n'existe entre les labels `usualTitle` et `author`. Cette dépendance était toutefois présente dans le cas de la composition séquentielle, alors que celle-ci n'a pas de sens.

L'annotation avec la composition hiérarchique se fait donc en suivant la hiérarchie des labels et combine les méthodes de composition parallèle et séquentielle. À chaque label l ayant un ou plusieurs fils dans la hiérarchie correspond une sous-partie de l'alphabet des label $\mathcal{Y}$ que l'on note $\mathcal{Y}_l$. $\mathcal{Y}_l$ est composé des labels qui sont fils de l dans la hiérarchie : $\mathcal{Y}_l = \{l' \mid l' \in \text{Fils}(l)\}$. À chaque niveau de l'arbre, la composition parallèle est utilisée. C'est-à-dire, pour une profondeur p fixée de la hiérarchie et pour tous les labels l de profondeur p, les champs aléatoires conditionnels $\mathcal{C}_l$ dont l'alphabet des labels est $\mathcal{Y}_l \cup \perp$ sont appliqués en parallèle. La composition séquentielle intervient au niveau de la récursion en profondeur dans la hiérarchie des labels. En effet, pour un label l de la hiérarchie, l'annotation avec les labels $\mathcal{Y}_l$ est effectuée après les annotations avec $\mathcal{Y}_{l'}$ pour tous les l' qui sont des ancêtres de l. Ces annotations sont donc connues et les dépendances dirigées longue distance que la composition séquentielle permet de simuler interviennent.

Algorithmes

Nous présentons dans un premier temps l'algorithme d'apprentissage pour la composition hiérarchique (algorithme 6). Celui-ci est récursif (comme l'est le parcours d'un arbre) et est défini par une fonction **HiTrain**. Cette fonction effectue l'apprentissage d'un champ aléatoire conditionnel pour une étape l. Elle prend en paramètres un échantillon d'apprentissage S composé de couples $(\mathbf{x}^j, \mathbf{y}^j)$, $\mathbf{x}^j$ et $\mathbf{y}^j$ étant définis respectivement sur $\mathcal{X}$ et $\mathcal{Y}$, la hiérarchie des labels, un label l ainsi que la fonction Gen de génération de fonctions de caractéristiques. Si le label l passé en paramètres est une feuille dans la hiérarchie de labels, aucun apprentissage n'est effectué. Dans le cas contraire, l'algorithme commence par générer l'ensemble d'apprentissage S_l, qui est constitué de couples $(\mathbf{x}_l^j, \mathbf{y}_l^j)$

définis respectivement sur $\mathcal{X}_l$ et $\mathcal{Y}_l \cup \{\perp\}$. $\mathcal{X}_l$ est l'ensemble des couples dont la première composante est l'observation $\mathbf{x}$ et la deuxième composante est l'annotation effectuée aux étapes précédentes : $\mathcal{X}_l = \mathcal{X} \times \bigcup_{l' \in \text{Anc}(l)} \mathcal{Y}_{l'} \cup \{\perp\}$. Si l'on considère un couple défini sur $\mathcal{X} \times \mathcal{Y}$, la fonction π'_l génère l'observation correspondant à ce couple à l'étape l de l'algorithme :

$$\pi'_l(x, y) = \begin{cases} (x, y) \text{ si } y \in \mathcal{Y}_{l'} \text{ tel que } l' \in \text{Anc}(l) \\ (x, \perp) \text{ sinon} \end{cases}$$

En étendant cette fonction aux vecteurs, l'ensemble d'apprentissage à l'étape l est défini de la façon suivante : $S_l = \{(\pi'_l(\mathbf{x}, \mathbf{y}), \text{Cpl}_l(\mathbf{y})) \mid (\mathbf{x}, \mathbf{y}) \in S\}$. Par exemple, dans le cas du corpus "Movie", on a les alphabets d'entrée suivants :

$$\begin{aligned} \mathcal{X}_{\texttt{movie}} &= \mathcal{X} \times \{\perp\} \\ \mathcal{X}_{\texttt{presentation}} &= \mathcal{X} \times \mathcal{Y}_{\texttt{movie}} \cup \{\perp\} \\ \mathcal{X}_{\texttt{directors}} &= \mathcal{X} \times \mathcal{Y}_{\texttt{movie}} \cup \mathcal{Y}_{\texttt{presentation}} \cup \{\perp\} \\ &\dots \end{aligned}$$

L'algorithme génère ensuite les fonctions de caractéristiques sur cet ensemble d'apprentissage S_l et ajoute des contraintes locales similaires à celles utilisées lors de la composition séquentielle :

$$\forall n \in \mathcal{C}_1, \ \Big(x_n = (x, y) \text{ tel que } y \neq \perp\Big) \Rightarrow \forall l' \in \mathcal{Y}_l, \ l' \notin \mathcal{S}_n$$

Ces contraintes imposent qu'à l'étape l, seuls les nœuds de l'entrée dont la deuxième composante est $\perp$ peuvent être annotés par un label appartenant à $\mathcal{Y}_l$. Le champ aléatoire conditionnel $\mathcal{C}_l$ est alors appris et ajouté à l'ensemble $\mathcal{SC}$ des champs aléatoires conditionnels appris. Enfin, cette fonction est appelée récursivement sur tous les labels l' fils de l dans la hiérarchie. Les résultats de ces appels récursifs sont aussi ajoutés à $\mathcal{SC}$. Enfin, la fonction **HiTrain** retourne l'ensemble $\mathcal{SC}$ de tous les champs aléatoires conditionnels appris. Afin d'effectuer l'apprentissage pour tous les labels de la hiérarchie, la fonction **HiTrain** est appelée sur la racine ϵ de cette hiérarchie.

Afin de parcourir la hiérarchie des labels, et donc les différents champs aléatoires conditionnels associés, l'algorithme d'annotation 7 est lui aussi récursif et est défini par la fonction **HiLabel**. Cette fonction prend en paramètres une observation $\mathbf{x}$ définie sur $\mathcal{X}_l$, la hiérarchie de labels ainsi qu'un label l correspondant à la position courante dans cette hiérarchie. Si le label l est une feuille de la hiérarchie des labels, aucune annotation n'est effectuée et la fonction retourne l'ensemble vide. Dans le cas contraire, la fonction commence par obtenir le champ aléatoire conditionnel $\mathcal{C}_l = (F_l, \Lambda_l)$ correspondant à l et dont l'alphabet des labels est $\mathcal{Y}_l \cup \perp$. $\mathcal{C}_l$ est ensuite utilisé pour annoter $\mathbf{x}$ et ainsi obtenir l'annotation y_l. Celle-ci est ajoutée à l'ensemble $\mathcal{A}$ des annotations. Ensuite, afin de pouvoir effectuer les annotations pour les étapes l' telles que $l' \in \text{Fils}(l)$, il est nécessaire de constituer la nouvelle observation définie sur $\mathcal{X}_{l'}$. Il est intéressant de noter que, pour tous les $l' \in \text{Fils}(l)$, on obtient le même $\mathcal{X}_{l'}$. En effet, par définition, $\mathcal{X}_l = \mathcal{X} \times \bigcup_{l' \in \text{Anc}(l)} \mathcal{Y}_{l'} \cup \{\perp\}$. Ainsi, si deux labels l' et l'' sont frères dans la hiérarchie, on a $\text{Anc}(l') = \text{Anc}(l'')$ et donc $\mathcal{X}_{l'} = \mathcal{X}_{l''}$. Il n'est donc nécessaire de générer qu'une seule fois l'observation définie sur $\mathcal{X}_{l'}$

Algorithme 6 Apprentissage pour la composition hiérarchique.

Fonction HiTrain(S, h, l, Gen) **:**

Entrée: un échantillon d'apprentissage S de couples $\{(\mathbf{x}^j, \mathbf{y}^j)\}_{j=1}^{j=m}$, la hiérarchie h de labels, un label l de cette hiérarchie, et la procédure de génération de fonctions de caractéristiques Gen.

1: $\mathcal{SC} = \emptyset$ `# l'ensemble des CRFs appris`
2: **si** l n'est pas une feuille **alors**
3: générer l'ensemble d'apprentissage S_l : $S_l = \{(\pi'_l(\mathbf{x}, \mathbf{y}), \text{Cpl}_l(\mathbf{y})) \mid (\mathbf{x}, \mathbf{y}) \in S\}$
4: $F_l = \text{Gen}(S_l)$
5: ajouter les contraintes locales
6: Ajouter le résultat de TrainCRF(S_l, F_l) à $\mathcal{SC}$
7: **pour** tous les $l' \in \mathcal{Y}_l$ **do**
8: Ajouter le résultat de **HiTrain**(S, l', Gen) à $\mathcal{SC}$
9: **fin pour**
10: **fin si**

Renvoie $\mathcal{SC}$

Main:

Entrée: un échantillon d'apprentissage S de couples $\{(\mathbf{x}^j, \mathbf{y}^j)\}_{j=1}^{j=m}$, la hiérarchie h de labels et la procédure de génération de fonctions de caractéristiques Gen.

Sortie: **HiTrain**$(S, \epsilon, \text{Gen})$

pour tous les $l' \in \text{Fils}(l)$. Pour cela, on définit la fonction ρ_l. Celle-ci prend en paramètre un couple (x, y), où x est défini sur $\mathcal{X}_l$ (x est donc aussi un couple) et y est défini sur $\mathcal{Y}_l \cup \{\perp\}$. La fonction ρ_l retourne alors l'observation définie sur $\mathcal{X}_{l'}$:

$$\rho_l((x_1, x_2), y) = \begin{cases} (x_1, y) \text{ si } y \in \mathcal{Y}_l \\ (x_1, x_2) \text{ sinon } (y = \perp) \end{cases}$$

Si l'on considère l'arbre de la figure 5.5, le nœud devant être annoté par `presentation` est, aux étapes ϵ, `movie` et `presentation` représenté par le couple (`NODE`, $\perp$). Par contre, à l'étape `directors` qui intervient après, il est représenté par le couple (`NODE`, `presentation`). En étendant la fonction ρ_l aux vecteurs comme nous l'avons fait pour la fonction π'_l, on obtient le calcul de l'observation complète (c'est-à-dire pour l'ensemble des nœuds de l'arbre) pour les étapes suivantes : $\mathbf{x}' = \rho_l(\mathbf{x}, \mathbf{y}_l)$.

Enfin, la fonction **HiLabel** est appelée récursivement pour tous les labels l' appartenant aux fils du label l. Les annotations résultant de ces appels récursifs sont ajoutées à l'ensemble $\mathcal{A}$ des annotations. En appelant la fonction **HiLabel** sur la racine ϵ de la hiérarchie des labels, on obtient en résultat l'ensemble $\mathcal{A}$ des annotations effectuées par les champs aléatoires conditionnels $\mathcal{C}_l$ pour tous les labels l qui ne sont pas des feuilles dans la hiérarchie. Pour obtenir l'annotation finale définie sur l'alphabet des labels complet $\mathcal{Y}$, on applique la fonction de combinaison Combine définie précédemment (*cf.* section 5.2.1.0), consistant à choisir l'annotation dont la probabilité marginale est la plus élevée, sur toutes les annotations de l'ensemble $\mathcal{A}$. L'utilisation de cette méthode est nécessaire

pour résoudre les éventuels conflits de même type que ceux rencontrés lors de la combinaison parallèle. En effet, la composition hiérarchique est une combinaison de la composition parallèle et de la composition séquentielle. Ainsi, si l'utilisation de contraintes permet d'éviter les conflits entre labels d'une même branche de la hiérarchie, par exemple entre les labels de $\mathcal{Y}_{\texttt{movie}}$ et ceux de $\mathcal{Y}_{\texttt{presentation}}$, il peut tout de même y avoir un conflit entre deux labels provenant de deux branches différentes. Par exemple, les annotations parallèles avec les alphabets $\mathcal{Y}_{\texttt{presentation}}$ et $\mathcal{Y}_{\texttt{rating}}$ peuvent occasionner un conflit.

Algorithme 7 Annotation pour la composition hiérarchique.

Fonction HiLabel$(\mathbf{x}, h, l)$
Entrée: une observation $\mathbf{x}$, la hiérarchie de labels h, un label l de cette hiérarchie
$\mathcal{A} = \emptyset$
si l n'est pas une feuille **alors**
  obtenir le champ aléatoire conditionnel $\mathcal{C}_l = (F_l, \Lambda_l)$ correspondant à l'étape l
  $\mathbf{y}_l = LabelCRF(\mathbf{x}, \mathcal{C}_l)$
  Ajouter $\mathbf{y}_l$ à $\mathcal{A}$
  intégrer $\mathbf{y}_l$ à $\mathbf{x} : \mathbf{x}' \leftarrow \rho_l(\mathbf{x}, \mathbf{y}_l)$
  pour tous les $l' \in \mathcal{Y}_l$ **do**
    Ajouter le résultat de **HiLabel**$(\mathbf{x}', l')$ à $\mathcal{A}$
  fin pour
fin si
Renvoie $\mathcal{A}$

Main:
Entrée: une observation $\mathbf{x}$, la hiérarchie de labels h
$\hat{\mathbf{y}} = \text{Combine}(\textbf{HiLabel}(\mathbf{x}, h, \epsilon))$
Sortie: $\hat{\mathbf{y}}$

Complexité

Du point de vue de la complexité, cette méthode de composition hiérarchique est assez similaire à la méthode de composition séquentielle. En effet, les différentes étapes de l'algorithme d'apprentissage peuvent ici aussi être effectuées en parallèle. En supposant une hiérarchie des labels d'arité a fixe, toutes les sous-parties de l'alphabet des labels sont donc de taille a et la complexité de l'algorithme d'apprentissage pour la composition hiérarchique est donc de $\mathcal{O}(G \times N \times a^K)$, où K est la taille des cliques maximales. Dans le cas le plus courant où la hiérarchie n'est pas d'arité fixe, a est la valeur de l'arité maximale de la hiérarchie. Concernant l'algorithme d'annotation, la complexité est légèrement plus importante. En effet, si à une profondeur donnée de la hiérarchie, les différentes étapes d'annotations peuvent être effectuées en parallèle, il n'en est pas de même pour les différentes profondeurs, celles-ci nécessitant les résultats des annotations précédentes. Ainsi, en notant p la profondeur et a l'arité maximales de la hiérarchie de labels, la complexité de l'algorithme d'annotation pour la composition hiérarchique est donc $\mathcal{O}(N \times p \times a^K)$.

5.2.3 Comparaison des trois méthodes de composition de CRFs

Nous effectuons maintenant un bilan sur les différentes méthodes de composition de champs aléatoires conditionnels que nous venons de présenter. Pour cela, nous nous appuyons essentiellement sur les dépendances entre les labels qu'il est possible de représenter avec ces trois méthodes en les comparant avec celles présentes dans les champs aléatoires conditionnels traditionnels.

Dans un premier temps, nous considérons les cas où l'utilisation de méthodes de composition fait perdre des dépendances. Les trois méthodes de composition, puisqu'elles sont des méthodes approchées, souffrent de ce problème. En effet, ceci est nécessaire pour gagner en complexité et rendre praticables des tâches qui ne l'étaient pas sans utiliser ces méthodes en raison de la taille de l'alphabet des labels. Toutefois, ces pertes diffèrent selon les méthodes. Les méthodes de composition parallèle et séquentielle occasionnent toutes les deux la perte des dépendances entre les labels n'appartenant pas à une même sous-partie de la partition de l'alphabet des labels. Ceci est alors fortement dépendant du choix de la partition. Toutefois, grâce aux dépendances dirigées longue distance que permet de simuler la composition séquentielle, ces pertes sont en partie compensées. Par exemple, dans le cas de l'annotation du corpus "Movie", les labels `presentation` et `tagLine` appartiennent respectivement à deux sous-parties différentes $\mathcal{Y}_2$ et $\mathcal{Y}_3$ de la partition. Ainsi, la dépendance entre ces deux labels est perdue avec la composition parallèle, tandis que, dans le cas de la composition séquentielle, la relation d'ordre définie sur la partition précise que $\mathcal{Y}_3$ est située après $\mathcal{Y}_2$. Le choix du label `tagLine` peut alors dépendre du choix du label `presentation` (mais pas l'inverse). De la même façon, avec la composition hiérarchique, on observe des pertes de dépendances entre certains labels. Toutefois, ces pertes sont moins problématiques car elles dépendent de la hiérarchie des labels et non d'une partition potentiellement arbitraire de l'alphabet des labels. De plus, comme dans le cas de la composition séquentielle, des dépendances dirigées viennent partiellement compenser ces pertes.

À l'exception de la composition parallèle qui n'offre qu'un pur gain de complexité algorithmique, et qui est donc à réserver à des tâches dans lesquelles les dépendances entre labels sont très limitées ou quand l'ensemble des labels est clairement séparable en sous-parties complètement indépendantes, ces méthodes de composition permettent aussi d'exprimer de nouvelles dépendances. En effet, les méthodes de composition séquentielle et hiérarchique simulent des dépendances longue distance dirigées. Ces nouvelles dépendances sont dues au fait que les annotations sont effectuées les unes à la suite des autres et qu'il est donc possible de considérer les résultats des annotations précédentes comme faisant partie de l'observation. On pourra tout de même distinguer ces deux méthodes par le fait que dans le cas de la composition séquentielle ces dépendances dépendent fortement du choix de la partition de l'alphabet des labels mais aussi de l'ordre défini sur cette partition, alors que ces dépendances sont introduites de façon naturelle avec la méthode composition hiérarchique.

5.2.4 Travaux associés

Nous présentons maintenant quelques travaux existants en relation avec les méthodes de composition de champs aléatoires conditionnels que nous venons de décrire. D'une part, les travaux de [Sutton and McCallum, 2005] considèrent, essentiellement dans le cadre de l'annotation de séquence, le cas où l'on souhaite réaliser plusieurs annotation en même temps, comme par exemple, dans le domaine du traitement des langues, annoter avec les labels *Part-Of-Speech*, puis identifier les entités nommées. Effectuer ces deux annotations d'une même séquence simultanément permet d'éviter l'accumulation d'erreurs commises lorsque celles-ci sont effectuées l'une après l'autre, les labels des deux tâches d'annotations pouvant influer l'une sur l'autre (dans les deux sens). Toutefois, effectuer plusieurs tâches d'annotations de ce type augmente significativement la taille de l'alphabet des labels et entraîne ainsi des temps d'apprentissage prohibitifs. Pour éviter cela, [Sutton and McCallum, 2005] proposent une méthode de composition de champs aléatoires conditionnels appelée *joint testing with cascaded training* dans laquelle plusieurs CRFs sont appris séquentiellement pour chaque tâche d'annotation, de façon plus ou moins similaire à notre méthode de composition séquentielle, puis les CRFs appris sont regroupés en un unique CRF pour l'annotation. Si, comme dans le cas de la composition séquentielle, cette méthode intègre les résultats de l'annotation précédente dans l'observation, elle ne permet toutefois pas de simuler des dépendances longues distances. En effet, l'utilisation des annotations précédentes est strictement restreinte au niveau des cliques, afin de garantir l'intégrité du modèle de dépendances lors de la recomposition des CRFs pour l'annotation.

D'autre part, les travaux de [Cohn et al., 2005] sont plus proches des nôtres. En effet, contrairement à ceux de [Sutton and McCallum, 2005] dont l'objectif final est d'effectuer plusieurs tâches d'annotation simultanément, ces travaux ont le même but de réduire la complexité des champs aléatoires conditionnels de façon à les adapter à des tâches réelles dont l'alphabet des labels est grand. Dans ce but, ils utilisent les codes correcteurs d'erreurs (*Error Correcting Output Code*) pour représenter les labels. Ainsi, chaque label de l'alphabet est représenté par un code binaire de longueur L, et la tâche d'annotation multi-labels peut alors être approximée par L tâches d'annotation binaires. Cette méthode s'inspire des travaux concernant l'utilisation des codes correcteurs d'erreurs dans le cadre de la classification multi-classes [Berger, 1999]. Étant donné un code de longueur L pour l'ensemble de l'alphabet des labels $\mathcal{Y}$, L champs aléatoires conditionnels binaires correspondant chacun à un bit du code binaire des labels peuvent alors être appris et appliqués en parallèle. L'intérêt principal de cette méthode est qu'elle réduit considérablement la complexité en limitant le nombre de labels à chaque étape à 2. Ainsi, la complexité en temps de l'algorithme d'annotation est de $\mathcal{O}(N \times L \times 2^K)$, où K est la taille des cliques maximales. Toutefois, [Cohn et al., 2005] précisent que le choix de la taille L du code binaire influence grandement la qualité des résultats. Par exemple, pour une tâche d'annotation *Part-Of-Speech* comportant 45 labels, il leur a été nécessaire d'utiliser un code de longueur 200 pour obtenir les meilleurs résultats. Quand on compare cette méthode de composition aux nôtres, on remarque tout d'abord que du point de vue de la complexité, celle-ci est a priori meilleure. Toutefois, l'utilisation d'un codage binaire rend l'interprétation des dépendances entre les labels dans un CRF beaucoup plus difficile. De plus, il n'est

pas non plus possible dans cette méthode de simuler de nouvelles dépendances dirigées entre les labels.

5.2.5 Évaluation empirique des trois méthodes de composition

Nous évaluons maintenant les trois méthodes de composition de champs aléatoires conditionnels que nous avons définies. Pour cela, nous effectuons deux séries d'expériences. Dans un premier temps, nous évaluons l'influence du choix de la partition de l'alphabet des labels sur les résultats obtenus avec les méthodes de composition parallèle et séquentielle. Ensuite, une seconde série d'expériences va nous servir à comparer empiriquement les trois méthodes de composition de champs aléatoires conditionnels que nous avons proposées. Le but de cette évaluation est double. Nous souhaitons montrer d'une part que ces méthodes permettent d'obtenir des résultats au moins comparables à ceux obtenus avec des champs aléatoires conditionnels sans composition, tout en réduisant considérablement le temps de calcul. Mais cette seconde série d'expériences va aussi nous permettre de comparer nos trois méthodes entre elles et mettre en évidence, par exemple, l'intérêt des dépendances longue distance apportées par les méthodes de composition séquentielle et hiérarchique.

Pour toutes ces expériences, nous considérons les tâches d'annotation des arbres XML des corpus "Courses" et "Movie". Afin d'évaluer nos résultats, nous mesurons une fois encore la F_1-mesure au niveau des labels. Nous évaluons aussi le gain de temps apporté par chacune de ces méthodes. Pour cela, nous mesurons le temps nécessaire à l'apprentissage des paramètres rapporté au nombre d'exemples dans l'échantillon d'apprentissage, ainsi que le temps moyen d'annotation d'un document du corpus.

Comparaison de différentes partitions de l'alphabet des labels

Nous commençons par évaluer l'influence sur les résultats du choix de la partition de l'alphabet des labels dans les méthodes de composition parallèle et séquentielle. Pour cela, nous allons comparer, sur le corpus "Movie", les résultats obtenus par les méthodes de composition parallèle et séquentielle avec une partition de l'alphabet des labels tirée aléatoirement et avec une partition choisie plus intelligemment. Le choix intelligent d'une partition de l'alphabet des labels est guidé par la DTD de sortie de cette tâche, représentée sur la figure 5.4. À l'aide de cette DTD, nous créons une sous-partie de l'alphabet des labels pour chaque fils de la racine. Ces sous-parties sont constituées de tous les labels pouvant apparaître sous le fils considéré. Une sous-partie supplémentaire ne comportant que le label correspondant à la racine doit aussi être créée.

Le tableau 5.5 présente les résultats obtenus par les deux méthodes de composition selon le choix de la partition de l'alphabet des labels. Ces chiffres sont le résultat d'expériences dans lesquelles nous avons fait varier le nombre d'exemples en apprentissage de 5 à 50, soit environ 5% du corpus. Nous utilisons ici les paramètres obtenant les meilleurs résultats en moyenne lors de l'utilisation des champs aléatoires conditionnels traditionnels. Ainsi, le modèle de dépendances choisi dans ces expériences est celui des 3-CRFs. De plus, nous avons fixé le paramètre de voisinage à 3. La mesure choisie ici est encore la macro moyenne des F_1-mesures de chaque label.

Nombre d'exemples	Composition Parallèle		Composition Séquentielle	
	Aléatoire	Intelligente	Aléatoire	Intelligente
5	95.94	97.06	83.05	**99.60**
10	98.39	99.89	84.30	**99.90**
20	98.81	99.41	79.01	**99.73**
50	99.24	99.58	79.80	**100**

TAB. 5.5 – Résultats des méthodes de composition parallèle et séquentielle selon le choix de la partition de $\mathcal{Y}$.

Ces premiers résultats montrent l'importance du choix de la partition. En effet, si la différence de performances n'est pas significative en utilisant la composition parallèle, la chute des performances est nettement plus forte avec la composition séquentielle. Cette chute s'explique par le fait qu'avec la partition aléatoire, les labels appartenant à une même sous-partie n'ont pas forcément de dépendances entre eux, tandis que certains labels fortement dépendants se retrouvent isolés dans des sous-parties différentes. Ces dépendances ainsi cassées empêchent donc les champs aléatoires conditionnels de correctement prédire l'annotation. De plus, dans le cas de la composition séquentielle, on a non seulement des dépendances locales au sein des sous-parties de la partition, mais aussi, selon l'ordre défini sur la partition, les dépendances longue distance dirigées simulées entre les sous-parties. Toutefois, avec la partition aléatoire, l'ordre de ces sous-parties perd son sens, et ces nouvelles dépendances ne peuvent pas venir compenser la perte de performances due à la partition de l'alphabet des labels.

Expériences sur les tâches d'annotation

Dans cette deuxième série d'expériences, nous comparons à la fois les trois méthodes de composition de champs aléatoires conditionnels que nous avons introduites dans ce chapitre entre elles, mais nous les comparons aussi aux champs aléatoires conditionnels traditionnels, c'est-à-dire sans effectuer de composition. Pour cela, nous les appliquons dans le cadre des tâches d'annotation des corpus "Courses" et "Movie". Nous faisons varier le nombre d'exemples dans l'ensemble d'apprentissage de la même façon que dans les expériences sans composition présentées dans la section 4.3.3 afin de comparer les résultats obtenus. Nous ne faisons toutefois pas varier le paramètre de voisinage et le fixons à 3, celui-ci étant le paramètre qui donnait les meilleurs résultats dans les expériences sans composition. Dans le cas de la composition parallèle et de la composition séquentielle, la partition utilisée est la partition dite "intelligente" présentée dans l'expérience précédente, tandis que la hiérarchie de labels considérée pour la composition hiérarchique est la DTD des arbres XML de sortie.

Les résultats de ces expériences sur le corpus "Movie" sont présentés dans le tableau 5.6. La première observation que l'on peut faire est que les 3 méthodes de composition offrent de très bonnes performances, comparables, voire même souvent supérieures, à celles obtenues avec les champs aléatoires conditionnels sans composition, que ce soit dans le cas des 2-CRFs ou des 3-CRFs. Toutefois, on remarque que la composition hiérarchique obtient des résultats décevants avec seulement 5 exemples dans l'ensemble d'apprentissage.

nombre d'exemples	2-CRFs				3-CRFs			
	normal	paral.	séq.	hiér.	normal	paral.	séq.	hiér.
5	98.82	98.83	99.48	89.81	98.67	97.06	**99.60**	88.52
10	98.55	99.59	**99.99**	99.98	98.43	99.89	99.90	99.68
20	97.65	99.66	99.83	**99.88**	98.70	99.41	99.73	99.28
50	99.66	99.89	**100**	99.97	99.72	99.58	**100**	99.60

TAB. 5.6 – F_1-mesure des méthodes de composition sur le corpus "Movie". La colonne "normal" présente les résultats des CRFs sans utiliser de méthode de composition.

nombre d'exemples	2-CRFs				3-CRFs			
	normal	paral.	séq.	hiér.	normal	paral.	séq.	hiér.
5	**90.36**	67.92	79.56	50.67	87.70	64.90	82.55	66.25
10	93.98	90.72	90.87	96.26	**97.81**	94.38	94.12	95.16
20	95.73	95.87	95.49	**99.30**	98.31	94.15	95.45	97.33
50	95.99	92.97	95.67	99.85	98.69	97.60	98.90	**99.95**

TAB. 5.7 – F_1-mesure des méthodes de composition sur le corpus "Courses". La colonne "normal" présente les CRFs sans utiliser de méthode de composition.

Ce comportement est corrigé à partir de 10 documents dans l'ensemble d'apprentissage. Enfin, on remarque, dans l'ensemble, que les meilleurs résultats sont obtenus avec les compositions séquentielles et hiérarchiques. Il est d'ailleurs intéressant de noter qu'à deux reprises, avec les 2-CRFs et les 3-CRFs, la composition séquentielle permet même d'obtenir une F-mesure de 100% quand 50 exemples ont été utilisés en apprentissage.

Nous évaluons maintenant nos trois méthodes de composition sur le corpus "Courses". Bien que la taille de l'alphabet des labels soit raisonnable pour ce corpus et ne nécessite a priori pas d'utiliser de telles méthodes de composition, la difficulté supérieure de ce corpus nous permet de tester leur robustesse. Les résultats obtenus sont présentés dans le tableau 5.7. On observe deux comportements différents. D'une part, les méthodes de composition parallèle et séquentielle obtiennent toutes deux des résultats inférieurs à ceux obtenus par les champs aléatoires conditionnels sans composition. Cela montre une des faiblesses de ces méthodes de composition sur des tâches plus complexes. Toutefois, avec suffisamment d'exemples en apprentissage, on remarque qu'elles permettent tout de même d'obtenir des résultats suffisamment proches de ceux obtenus avec les champs aléatoires conditionnels sans composition. D'autre part, les résultats avec la composition hiérarchique montrent que, de la même façon que sur le corpus "Movie", cette méthode n'obtient pas de bons résultats avec très peu d'exemples en apprentissage. Toutefois, elle permet une fois de plus d'obtenir, étant donné un nombre suffisant d'exemples, de meilleurs résultats qu'avec les champs aléatoires conditionnels sans composition, atteignant une excellente F_1-mesure de 99.95%, tandis que sans composition, ce score n'était que de 98.69%.

Maintenant que nous avons montré que les méthodes de composition permettent d'obtenir des performances au moins comparables à celles obtenues sans composition, nous nous intéressons à la complexité de ces méthodes. En effet, leur objectif premier était de réduire considérablement la complexité des champs aléatoires conditionnels. Pour cela,

	normal	paral.	séq.	hiér.
Temps moyen apprentissage	383.35	10.4	9.85	12.14
Temps moyen annotation	1.98	0.68	1.43	1.34

TAB. 5.8 – Comparaison des temps de calcul sur le corpus "Movie" avec les 3-CRFs. La colonne "normal" correspond aux 3-CRFs sans méthode de composition.

nous mesurons les temps de calculs moyens des différentes méthodes afin de les comparer à ceux obtenus sans utiliser de méthode de composition. Les temps de calcul obtenus sur le corpus "Movie" sont présentés dans le tableau 5.8. Le temps de calcul en apprentissage est rapporté au nombre d'exemples de l'ensemble d'apprentissage, tandis le temps d'annotation est le temps moyen pour annoter un arbre XML.

Les résultats obtenus correspondent à nos attentes. En effet, quelle que soit la méthode de composition utilisée, à la fois l'annotation et l'apprentissage sont plus rapides, la méthode la plus rapide étant la composition parallèle. On notera tout de même que le gain est nettement plus important pour l'apprentissage que pour l'annotation. Ceci s'explique par le fait que, lorsque qu'une méthode de composition est utilisée, l'apprentissage de chaque champ aléatoire conditionnel nécessite généralement moins de pas de gradient, l'alphabet des labels étant plus petit. Il en résulte un gain de temps très significatif, l'apprentissage étant 30 à 40 fois plus rapide selon la méthode de composition utilisée. L'annotation, quant à elle, est tout de même en moyenne 2 fois plus rapide, avec un temps d'annotation raisonnable d'environ 1 seconde avec les méthodes de composition, contre 2 sans ces méthodes.

5.3 Conclusion

Nous avons donc présenté, dans ce chapitre, deux techniques efficaces d'amélioration de la complexité des champs aléatoires conditionnels que sont les contraintes et les méthodes de composition. Ces méthodes se distinguent essentiellement sur leur approche exacte ou approchée. En effet, d'un côté, l'introduction de contraintes dans les CRFs ne change pas les modèles de dépendances, mais impose simplement une probabilité conditionnelle $p(\mathbf{y}|\mathbf{x})$ nulle à toutes les annotations $\mathbf{y}$ ne respectant pas les contraintes. De l'autre côté, les méthodes de composition, afin de réduire plus significativement la complexité, brisent les dépendances entre certains labels en approximant un CRF défini sur l'alphabet des labels $\mathcal{Y}$ par plusieurs CRFs plus simples, définis sur des sous-parties de $\mathcal{Y}$.

De plus, nous avons montré empiriquement, sur des tâches d'annotation d'arbres, que ces deux techniques d'optimisation des champs aléatoires conditionnels ne se font pas au détriment des résultats obtenus. En effet, non seulement, ceux-ci sont comparables à ceux obtenus par les champs aléatoires conditionnels normaux, mais, dans certains cas, aussi bien les contraintes que les méthodes de composition permettent même d'améliorer ces résultats. Afin de confirmer les bons résultats obtenus par ces deux techniques, ainsi que par les champs aléatoires conditionnels pour les arbres XML en général, nous les appliquons dans le chapitre suivant à des tâches de transformation d'arbres XML à plus grande échelle.

Chapitre 6

Expériences en transformation d'arbres XML

Ce dernier chapitre est consacré aux applications sur lesquelles nous avons mis en œuvre les champs aléatoires conditionnels pour les arbres XML. Ces applications consistent toutes en des tâches de transformations d'arbres XML. Nous séparons ces tâches en deux catégories selon le type de documents XML en entrée. D'un côté, nous avons des tâches où les arbres XML d'entrée sont orientés données, c'est-à-dire des documents dans lesquels une grande partie de l'information est située dans la structure arborescente. Typiquement, ce type de documents XML possède peu d'informations au niveau des feuilles texte. Le deuxième type de tâches que nous abordons consiste en la transformation d'arbres XHTML en arbres XML. Contrairement aux arbres XML orientés données, les arbres XHTML sont orientés document. La structure interne ne donne pas d'informations sur la sémantique du contenu textuel, mais sur la mise en page de ce contenu. Ce type de document a pour sa part tendance à contenir beaucoup plus d'informations au niveau des feuilles.

Nous présentons donc tout d'abord deux tâches de transformations du XML vers le XML. La première est une tâche d'extraction d'informations structurées sur un corpus XML de descriptions de films, tandis que la seconde est une tâche de *Schema Matching* sur un corpus d'annonces immobilières et correspond à l'expérience utilisée par [Doan et al., 2001] pour évaluer le système LSD. Nous appliquons ensuite nos méthodes au cas des transformations d'arbres XHTML en arbres XML. Dans ce cadre, nous avons dans un premier temps participé à la tâche de *Structure Mapping* du challenge XML Mining 2006[12] [Denoyer and Gallinari, 2006]. Cette tâche consiste en la transformation de pages Web issues de sites de cinéma et décrivant un film en documents XML orientés données. Nous nous sommes ensuite évalués sur une tâche de génération automatique de flux RSS à partir de pages Web correspondant aux résultats d'une requête. Encouragé par les excellents résultats obtenus, nous avons mis en place une application en ligne permettant à un utilisateur d'apprendre de tels générateurs de flux RSS.

[12] `http://xmlmining.lip6.fr`

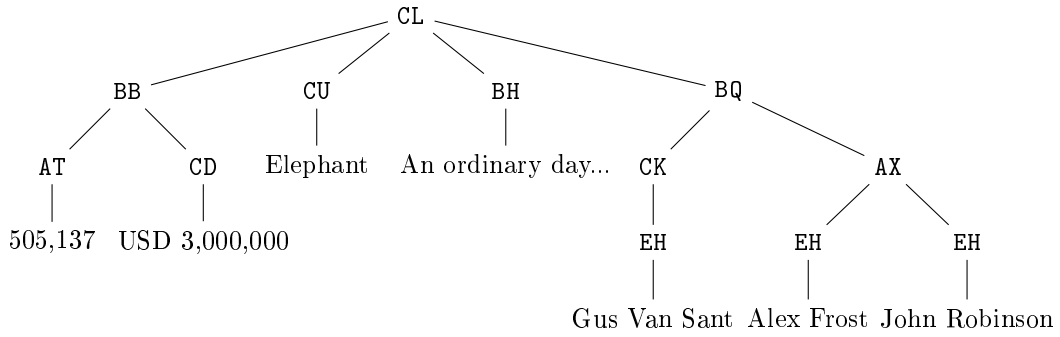

FIG. 6.1 – Exemple partiel d'arbre d'entrée dans le corpus "MovieDB".

6.1 Transformation du XML vers le XML

Nous présentons donc dans un premier temps nos travaux dans le domaine de la transformation d'arbres XML orientés données. Ces transformations prennent en entrée un arbre XML suivant un schéma source et le transforment en un arbre XML suivant un autre schéma appelé schéma cible. Dans les deux expériences qui suivent, à la fois l'arbre d'entrée et celui de sortie sont des arbres XML orientés données, c'est-à-dire des documents XML dans lesquels la structure interne de l'arbre apporte beaucoup d'informations sur le contenu textuel.

Nous présentons maintenant les résultats obtenus avec des champs aléatoires conditionnels sur une expérience d'extraction d'informations structurées, avant de nous évaluer sur une tâche de *Schema Matching*.

6.1.1 Extraction d'informations structurées

Nous appliquons donc dans un premier temps les champs aléatoires conditionnels pour les arbres XML à une tâche d'extraction d'informations structurées. Contrairement aux tâches d'extraction d'informations classiques qui consistent à extraire des informations sous la forme de singletons ou de n-uplets, l'extraction d'informations structurées consiste pour sa part à extraire, dans les documents fournis en entrée, des informations sous la forme d'un arbre. Cette tâche correspond tout à fait à une tâche de transformation d'arbres.

Présentation des données

Pour cette tâche d'extraction d'informations structurées, nous utilisons le corpus "MovieDB". Ce corpus est constitué d'arbres XML fournissant chacun une description détaillée d'un film. Les documents de ce corpus sont fortement orientés données : le texte présent dans les feuilles est très court et les étiquettes des nœuds internes apportent beaucoup d'informations. Toutefois, la sémantique de ces étiquettes n'est pas connue a priori et doit être découverte pour obtenir de bons résultats. Par exemple, le titre du film peut apparaître dans un nœud étiqueté par `CU`. Il n'est en effet pas évident a priori que `CU` correspond au titre. De plus, ce corpus propose plusieurs schémas différents pour les arbres

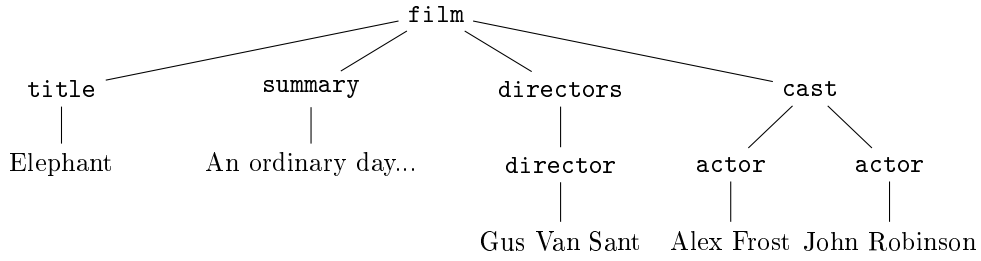

FIG. 6.2 – Exemple partiel d'arbre de sortie du corpus "MovieDB".

XML d'entrée (la sortie restant la même). Nous choisissons de ne nous intéresser ici qu'à un seul schéma d'entrée. Nous précisons néanmoins que les résultats obtenus avec les autres schémas sont comparables à ceux présentés ici. Un exemple d'arbre d'entrée suivant ce schéma est représenté sur la figure 6.1. Avec ce schéma d'entrée, nous utilisons une sous-partie de ce corpus constituée de 100 documents XML, chacun étant constitué en moyenne de 164 nœuds éléments et 591 nœuds texte. Les arbres de ce corpus sont peu profonds : presque tous les arbres sont de profondeur 4, mais sont en revanche très larges, avec une arité moyenne d'environ 65. Les arbres de sorties ont pour leur part une moyenne de 41 éléments et 193 feuilles texte. Un exemple de sortie est représenté en figure 6.2.

Modélisation de la tâche de transformation

Sur ce corpus, la tâche consiste donc à extraire des données suivant une DTD de sortie. Cette tâche est modélisée sous la forme d'une tâche d'annotation en utilisant les annotations de type "opérations d'édition". Seules les opérations de renommage et de suppression de nœuds sont ici utilisées. Ainsi, chaque nœud des documents XML d'entrée est annoté avec l'étiquette du nœud correspondant dans la DTD de sortie. Les nœuds n'ayant pas de correspondant dans la DTD de sortie sont annotés par le label `delete`, tandis que les feuilles texte à conserver sont annotées par $\perp$. Une des difficultés de ce corpus est qu'il nécessite de tenir compte du contexte. En effet, selon le contexte, deux nœuds du document d'entrée ayant la même étiquette peuvent être annotés différemment. Par exemple, comme le montre l'arbre de la figure 6.1, un nœud dont l'étiquette est `EH` peut correspondre soit au nom d'un réalisateur, soit à celui d'un acteur et peut donc être annoté soit par `director`, soit par `actor`. Dans ce cas, seule l'étiquette du père de ce nœud permet de distinguer les deux.

Pour annoter les arbres XML d'entrée de cette façon, nous utilisons le modèle des 3-CRFs, qui a prouvé dans les expériences précédentes qu'il permettait d'obtenir les meilleurs résultats dans la plupart des cas. De plus, l'alphabet des labels étant de taille raisonnable (22 labels en comptant $\perp$), nous n'utilisons dans cette première expérience aucune contrainte et aucune méthode de composition. De cette façon, les résultats qui suivent sont bruts et ne tirent pas partie de la connaissance préalable du schéma de sortie.

Label	Rappel	Précision	F_1-mesure
`actor`	97.01	100	98.48
`aspect ratio`	100	100	100
`budget`	72.06	84.48	77.78
`cast`	96.39	100	98.16
`company`	100	100	100
`countries`	99.49	100	99.74
`date`	100	99.63	99.81
`delete`	99.82	99.18	99.50
`director`	97.75	98.31	98.03
`directors`	97.75	98.58	98.17
`film`	100	99.01	99.50
`film format`	99.56	100	99.78
`language`	90.24	100	94.87
`languages`	96.50	100	98.22
`production`	100	100	100
`rating`	100	100	100
`release dates`	100	99.72	99.86
`summary`	98.26	100	99.12
`tagline`	92.19	100	95.93
`title`	100	100	100
`writing credits`	95.28	92.24	93.73
Macro Moyenne	96.78	98.63	97.65
Micro Moyenne	99.28	99.30	99.29

TAB. 6.1 – Résultats de l'annotation sur le corpus "MovieDB".

Résultats

Les résultats présentés ici sont la moyenne de 5 itérations. À chaque itération, un cinquième du corpus (soit 20 documents) a été utilisé en tant qu'ensemble d'apprentissage d'un champ aléatoire conditionnel, le reste des documents servant à l'évaluation du CRF appris. Ainsi, chaque document du corpus a été utilisé exactement une fois en apprentissage et quatre fois en test.

Nous présentons, dans un premier temps, sur le tableau 6.1 les résultats en termes d'annotation. Pour cela, rappel, précision et F_1-mesure sont calculés pour chaque label, et les macro et micro moyennes sur ces labels sont calculées. Ces résultats montrent une très bonne performance générale des CRFs sur cette tâche, la plupart des labels obtenant une F_1-mesure supérieure à 98%. De plus, si l'on observe les moyennes, on remarque que la micro moyenne de la F_1-mesure est supérieure à 99% et est surtout supérieure à la macro moyenne. Ce résultat montre que les labels apparaissant le plus souvent, c'est-à-dire ceux correspondant aux étiquettes les plus présentes dans les arbres XML de sortie, obtiennent les meilleurs scores. En effet, si l'on compare la fréquence d'apparition des étiquettes dans les arbres de sortie, on remarque que les labels obtenant les moins bons résultats sont ceux apparaissant le moins souvent. Par exemple, le label `budget` n'apparaît que dans 1

Chemins	Sous-arbres	Chemins + Sous-arbres
91.42	87.56	87.56

TAB. 6.2 – Évaluation des arbres XML obtenus pour la tâche d'extraction d'informations structurées sur le corpus "MovieDB".

document sur 5, tandis que `tagLine` et `writing_credits` n'apparaissent que dans environ 1 document sur 2. Ainsi, moins d'exemples concernant ces labels étaient présents dans les ensembles d'apprentissage, expliquant en grande partie les résultats légèrement en retrait sur ces labels.

Dans un second temps, la tâche consistant en la transformation d'arbres XML, nous évaluons la qualité des arbres obtenus après transformation, c'est-à-dire après application des opérations d'édition correspondant aux labels affectés aux nœuds. Dans ce but, le tableau 6.2 présentent les résultats en utilisant les méthodes de comparaison d'arbres présentées dans la section 2.2.3 qui calculent la F_1-mesure sur les ensembles de chemins, de sous-arbres et de paires composées du chemin et du sous-arbres. Ces trois mesures viennent confirmer les bons résultats augurés précédemment en évaluant l'annotation. En effet, dans plus de 91% des cas, les chemins de la racine aux feuilles sont corrects. De plus, les évaluations sur les sous-arbres, qui sont beaucoup plus strictes, montrent une F_1-mesure de plus de 87%, plusieurs documents obtenant même un score de 100%. Ces résultats constituent une première preuve de la qualité des transformations qu'il est possible d'effectuer avec les 3-CRFs.

6.1.2 Schema Matching

Nous abordons maintenant la tâche plus ardue du *Schema Matching*. Contrairement à la tâche précédente où tous les arbres XML d'entrée suivaient le même schéma, dans cette tâche, les arbres XML fournis en entrée proviennent de plusieurs sources différentes, chacune ayant son propre schéma. Le but de la tâche est d'apprendre à transformer ces documents d'entrée pour leur faire suivre un schéma commun appelé **schéma médiateur**.

Présentation des données

Pour évaluer les champs aléatoires conditionnels sur ce problème, nous utilisons le corpus "Real Estate I" constitué par Doan. Ce corpus contient 10000 documents d'environ 35 nœuds chacun, décrivant des annonces immobilières provenant de cinq sources différentes. À chaque source correspond un schéma différent, et un schéma médiateur constitué de 16 étiquettes distinctes est connu. Les documents XML de ce corpus sont, comme précédemment, orientés données, mais ils possèdent un contenu texte assez riche fournissant un grand nombre d'informations qu'il est nécessaire d'utiliser lors de l'annotation.

Modélisation de la tâche

Nous modélisons cette tâche de *Schema Matching* sous la forme d'une tâche d'annotation d'arbres XML en annotant chaque nœud de type élément des documents d'entrée

avec l'étiquette qui lui correspond dans le schéma médiateur. Lorsque le nœud n'a pas d'équivalent dans l'arbre de sortie, le label $\perp$ est utilisé. Cette modélisation ne correspond à aucune des deux méthodes d'annotation pour la transformation présentées dans cette thèse, mais se rapproche des méthodes classiques de *Schema Matching*, dans lesquelles les algorithmes recherchent des correspondances entre les éléments de plusieurs schémas.

Étant donné que le schéma de sortie est préalablement connu, nous avons choisi d'utiliser le modèle des 3-CRFs afin de profiter au maximum des dépendances existant possiblement entre les différents labels. Le choix des 3-CRFs nous a aussi permis d'intégrer aux champs aléatoires conditionnels les contraintes résultant de cette DTD.

Résultats

Les résultats qui suivent ont été obtenus en effectuant 20 expériences. Pour chaque expérience, trois sources sont utilisées pour l'apprentissage et les deux autres sources servent au test. Dans un premier temps, les 3 sources pour l'apprentissage sont choisies aléatoirement. L'échantillon d'apprentissage est alors constitué en tirant aléatoirement 5 exemples annotés de chacune de ces 3 sources, obtenant ainsi un total de 15 documents dans l'échantillon. Tous les documents suivant les 2 sources restantes sont utilisés pour tester le champ aléatoire conditionnel appris. Dans un premier temps, nous avons évalué les performances des 3-CRFs appris en évaluant la qualité de l'annotation des documents de test. Pour cela, nous avons mesuré précision, rappel et F_1-mesure pour chaque label, c'est-à-dire chaque étiquette de la DTD de sortie. Le tableau 6.3 montre que les 3-CRFs obtiennent d'excellents résultats sur cette tâche. En effet, la précision sur l'ensemble des labels atteint presque les 99% tandis que la F_1-mesure est de 92%, en utilisant la micro moyenne. Ces résultats montrent que les 3-CRFs ont réussi à la fois à tirer parti des dépendances entre les labels mais aussi à utiliser les informations contenues dans le texte pour obtenir de très bonnes performances.

Les résultats du système LSD sur ce corpus présentés dans [Doan et al., 2001] suivent une méthode d'évaluation différente de la nôtre et directement liée à la tâche de *Schema Matching*. Leur objectif est d'identifier une fonction de l'ensemble des étiquettes des documents d'entrée vers l'ensemble des étiquettes du schéma médiateur. Dans un but de comparaison, nous reproduisons ces expériences dans les mêmes conditions que celles décrites dans [Doan et al., 2001]. Pour cela, pour chacune de nos 20 expériences, nous avons calculé une fonction de l'ensemble des étiquettes des schémas d'entrée vers l'ensemble des étiquettes du schéma médiateur à l'aide des documents annotés par les 3-CRFs. Cette fonction est obtenue en choisissant, pour chaque étiquette des schémas sources, le label qui lui est affecté le plus souvent. De la même façon que dans les 3-CRFs, le système LSD utilise à la fois l'étiquette et son contenu textuel pour apprendre les correspondances. Toutefois, contrairement aux 3-CRFs, la correspondance pour chaque étiquette est, avec LSD, apprise indépendamment, sans tenir compte de son contexte.

Nous évaluons maintenant notre fonction de correspondance en la comparant avec la fonction correcte en utilisant la mesure d'*accuracy* définie dans [Doan et al., 2001]. Cette mesure calcule la proportion d'étiquettes des schémas sources ayant un correspondant dans le schéma médiateur qui ont été correctement prédits. Les deux types d'erreur possibles sont donc l'association d'une étiquette d'un schéma source à une mauvaise étiquette du

Label	Précision	Rappel	F_1-mesure
agent_address	100	100	100
agent_fax	100	78.33	87.85
agent_name	100	98.58	99.28
agent_phone	95.97	81.73	88.28
bathrooms	100	75.84	86.26
bedrooms	100	97.65	98.81
firm_address	100	99.99	99.99
firm_name	99.80	93.91	96.77
firm_phone	90.81	95.77	93.23
house_address	100	90.24	94.87
garage	100	99.82	99.91
house_description	100	59.89	74.91
lot_area	99.99	99.31	99.65
mls_number	100	100	100
price	100	100	100
school	100	34.56	51.37
Macro Moyenne	98.95	79.57	88.21
Micro Moyenne	99.16	87.85	91.95

TAB. 6.3 – Résultats de l'annotation sur le corpus Real Estate I.

schéma médiateur, ou l'oubli d'une correspondance pour une étiquette ayant un équivalent dans le schéma médiateur. Les 3-CRFs obtiennent une *accuracy* de $96,4\%$, alors que le système LSD n'obtenait quant à lui qu'environ 80%. Nous expliquons essentiellement ce gain de performance par le fait que, contrairement à LSD, les 3-CRFs utilisent le contexte d'une étiquette pour trouver sa correspondance. En se servant ainsi de l'ensemble de la structure et pas uniquement de l'étiquette du nœud et de son contenu textuel, les 3-CRFs ont pu découvrir un plus grand nombre de correspondances correctes.

6.2 Transformation du XHTML vers le XML

Nous nous intéressons maintenant aux applications de transformations d'arbres XHTML en arbres XML. Si ce type de transformations est un cas particulier de la transformation du XML vers le XML, il en est tout de même fortement différent de par le type de données fournies en entrée. En effet, les arbres d'entrée sont ici des arbres XHTML, c'est-à-dire des arbres XML orientés document. Dans ces arbres XML, la structure donne des indications sur la façon de présenter les données et non sur leur nature, à l'inverse des documents XML orientés données dont la structure fournit une information sur le contenu textuel. Cette distinction peut rendre les tâches de transformation du XHTML vers le XML plus difficiles que les tâches précédentes.

La transformation du XHTML vers le XML possède de nombreux intérêts. En effet, les données présentes sur le Web suivent, pour la plupart, le format XHTML. Ainsi, être capable d'apprendre à transformer automatiquement des documents XHTML en documents

XML permet de réutiliser les données contenues dans ces documents. La réutilisation de ces données peut avoir pour but de les intégrer dans des bases de données. On rejoint ainsi le champ de l'extraction d'informations. Dans ce domaine, nous présentons les résultats obtenus avec des champs aléatoires conditionnels sur le challenge XML Mining. La transformation du XHTML vers le XML peut aussi avoir pour but de transformer la façon de présenter les données dans une pages XHTML, voire même de les résumer. Nous présentons ainsi une application dont le but est d'apprendre à générer automatiquement des flux RSS à partir de pages Web correspondant aux résultats d'une recherche.

6.2.1 Challenge XML Mining

Une première tâche de transformation du XHTML vers le XML sur laquelle nous avons testé les champs aléatoires conditionnels est la tâche de *Structure Mapping* du challenge XML Mining 2006[13] [Denoyer and Gallinari, 2006].

Présentation de la tâche et des données

La tâche de *Structure Mapping* du challenge XML Mining [Denoyer and Gallinari, 2006] consiste en la transformation d'arbres XHTML, en l'occurrence des pages Web, fournissant une description détaillée de films (titre, réalisateur, distribution, *etc.*) en des arbres XML contenant les mêmes données. Les arbres XHTML de ce challenge proviennent de sites de cinéma connus comme AllMovie[14], IMDB[15] (*Internet Movie DataBase*) et Allociné[16], tandis que les arbres XML de sortie sont issus de la base de données d'IMDB.

Dans les trois corpus, les arbres XHTML d'entrée sont de purs documents XHTML dans le sens où ils sont complètement orientés document, c'est-à-dire que la majeure partie des informations est située aux feuilles et la structure des documents n'apporte que des informations de présentation. Toutefois, celle-ci peut tout de même être utilisée pour identifier les informations. Parmi ces trois corpus, on distingue deux groupes : le corpus AllMovie, et les corpus Allociné et IMDB.

D'une part, pour chaque document du corpus AllMovie, deux versions différentes de l'arbre XHTML, appelées html1 et html2, sont disponibles. Ces deux versions décrivent les mêmes films, mais la première version ne contient que la partie du document XHTML correspondant à la description du film, tandis que la deuxième version contient aussi des informations inutiles, comme par exemple l'en-tête du document, ou encore des images. Les corpus d'apprentissage et de test sont presque de la même taille, avec respectivement 692 et 693 documents. Le nombre moyen de nœuds par arbres pour ces corpus est de 1100 pour html1 et 1250 pour html2. De plus, pour nous aider dans notre tâche de transformation, nous avions besoin de la DTD des arbres XML de sortie. Celle-ci n'étant pas fournie, nous l'avons inférée à l'aide de l'algorithme décrit dans [Bex et al., 2006]. Cette DTD de sortie est composée de 63 éléments parmi lesquels 39 contiennent de l'information textuelle.

[13] `http://xmlmining.lip6.fr`
[14] `http://www.allmovie.com`
[15] `http://www.imdb.com`
[16] `http://www.allocine.fr`

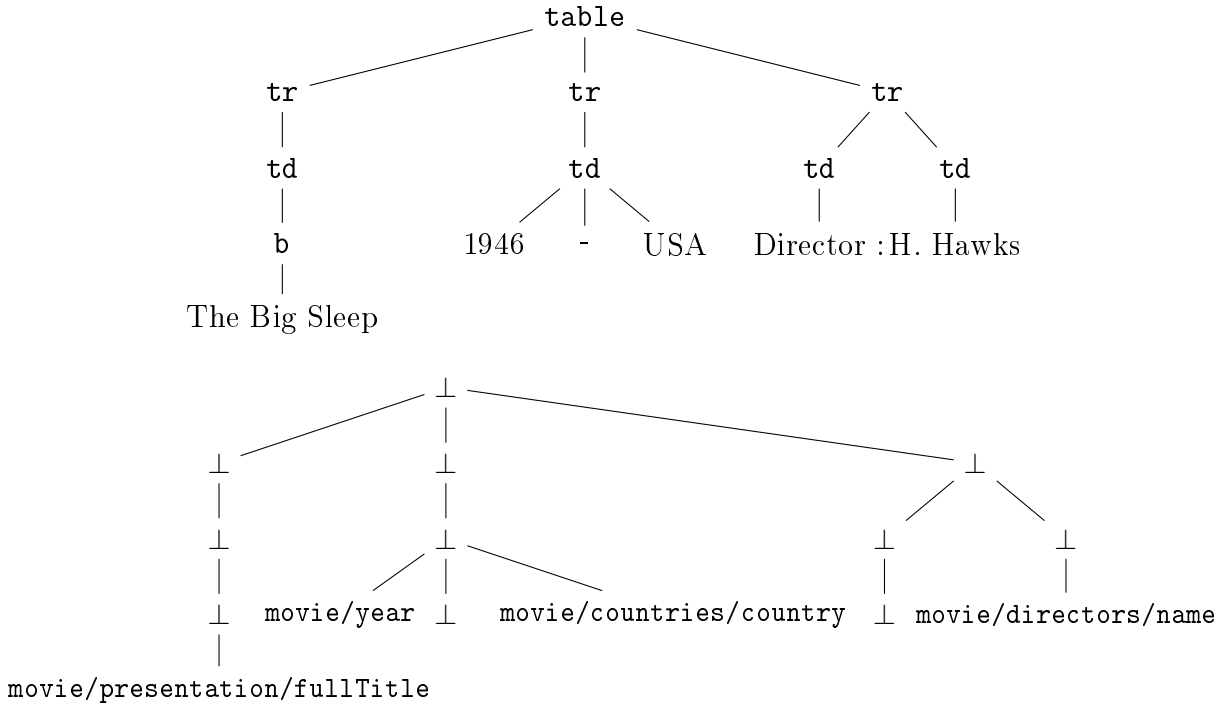

FIG. 6.3 – Exemple simplifié d'un arbre XHTML d'entrée (haut) du corpus "AllMovie" et de son annotation (bas) pour la tâche de *Structure Mapping* du challenge XML Mining.

Les corpus Allociné et IMDB sont eux composés d'environ 13000 documents. Les documents XHTML d'entrée sont d'une taille nettement inférieure aux documents XHTML du corpus AllMovie, avec une moyenne de 119 et 90 nœuds respectivement pour Allociné et IMDB. Les documents XML de sortie sont identiques pour ces deux corpus. La DTD de sortie, inférée à l'aide de l'algorithme décrit dans [Bex et al., 2006], comporte 55 éléments, parmi lesquels 22 contiennent de l'information textuelle. De plus, ces corpus possèdent la particularité de ne pas fournir dans les documents d'entrée, l'intégralité des informations contenues dans les documents XML de sortie. Ainsi, il nous est impossible d'obtenir des résultats parfaits (*ie.* 100%).

Les données XHTML à transformer étant orientées document, un problème se pose. En effet, dans ce type de documents XML, la majeure partie des informations se situe aux feuilles de l'arbre. Ainsi, il est possible qu'un unique nœud texte d'un document XHTML d'entrée contienne plusieurs informations différentes qui correspondent à plusieurs nœuds différents selon la DTD de sortie. Par exemple, dans le corpus XHTML d'entrée, la date de sortie, le format et la durée du film sont tous trois dans un unique nœud texte. Il est donc nécessaire, dans une phase de pré-traitement, de segmenter les nœuds texte, créant ainsi plusieurs nœuds consécutifs. Ce pré-traitement n'a été nécessaire que pour les deux versions du corpus AllMovie.

Modélisation de la transformation

Afin de modéliser cette transformation par une tâche d'annotation des arbres XHTML d'entrée, nous utilisons la méthode d'annotation par les chemins dans le schéma de sortie (*cf.* section 1.4.1), par exemple `movie/presentation/fullTitle` ou `movie/tagLine`. Les nœuds internes (de type élément) sont annotés par $\perp$. Si leur intérêt en termes d'annotation est alors réduit, ces nœuds internes, ainsi que toute la structure, sont exploités pour choisir correctement les labels associés aux feuilles. Dans le cas présent, le choix de ne pas utiliser la méthode d'annotation de type "opérations d'édition", et ainsi de ne pas annoter les nœuds internes est justifié par le fait que, sur ce corpus, la structure des documents d'entrée et celle des documents de sortie sont très différentes. Ainsi, avec ce type d'annotation, il serait nécessaire d'avoir des labels indiquant de renommer un nœud et d'en insérer un en même temps. Ceci entraînerait alors une explosion de la taille de l'alphabet des labels.

L'alphabet des labels est aussi complété par des labels indiquant qu'un nœud fait partie d'un ensemble de nœuds consécutifs correspondant à un seul nœud dans le document XML de sortie : par exemple, `movie/presentation/fullTitle_continued` ou `movie/synopsis_continued`. Ainsi, lorsque deux nœuds consécutifs d'un arbre XHTML d'entrée sont annotés respectivement par `movie/synopsis` et `movie/synopsis_continued`, le contenu textuel de ces deux nœuds est concaténé et constitue le contenu du nœud `synopsis`, fils de `movie`, dans l'arbre XML de sortie. Enfin, le label $\perp$ est utilisé pour annoter les nœuds internes ainsi que ceux qui n'apportent pas d'information. On obtient ainsi, pour le corpus AllMovie, un alphabet de 66 labels utiles, tandis que seulement 24 labels sont nécessaires dans le cas des corpus Allociné et IMDB, qui ne nécessite pas l'utilisation des labels de type "continued".

La figure 6.3 montre un exemple simplifié d'un arbre XHTML d'entrée et de son annotation. À partir d'un arbre XHTML annoté de la sorte, l'arbre XML de sortie est obtenu à l'aide d'un simple post-traitement, décrit dans la section 1.4.1, prenant en paramètres un arbre XHTML annoté et le schéma de sortie.

Choix du modèle

Afin d'apprendre à effectuer cette tâche de transformation via une tâche d'annotation d'arbres, nous utilisons le modèle des 3-CRFs. Toutefois, obtenir de bons résultats sur cette tâche passe par la nécessité d'avoir un très grand nombre de fonctions de caractéristiques (plus de 7000) définies sur un alphabet de 66 labels dans le cas des corpus AllMovie. De plus, les arbres XHTML à annoter possèdent chacun plus de 1100 nœuds une fois la segmentation décrite précédemment effectuée. Ces paramètres impliquent qu'une application naïve du modèle est impossible du point de vue de la complexité : $7000 \times 1100 \times 66^3 \approx 2.10^{12}$. Ainsi, pour rendre la tâche réalisable, nous utilisons les optimisations des champs aléatoires conditionnels que nous avons présentées dans le chapitre 5 : nous introduisons des contraintes et nous utilisons une méthode de composition de champs aléatoires conditionnels.

Sur cette application, il est possible de définir deux types de contraintes. Dans un premier temps, du fait que seuls les nœuds texte sont annotés, il est possible d'ajouter

une contrainte locale sur tous les autres types de nœuds empêchant l'annotation par tout label autre que $\perp$. De plus, il est aussi nécessaire d'ajouter des contraintes globales pour les labels de type "continued" correspondant aux ensembles de nœuds consécutifs représentant une unique information. En effet, annoter un nœud avec le label `synopsis_continued` n'a de sens que si le label de son frère gauche est `synopsis` ou `synopsis_continued`. Ainsi, pour chaque label de type "continued", nous ajoutons des contraintes uniformes du type :

$$\forall c \in \mathcal{C}_3,\ \forall y_1 \in \mathcal{Y}, y_2 \in \mathcal{Y} \setminus \{\texttt{synopsis}, \texttt{synopsis\_continued}\} \\ (y_1, y_2, \texttt{synopsis\_continued}) \notin \mathcal{S}_c$$

où $\mathcal{C}_3$ est l'ensemble des cliques triangulaires du graphe d'indépendances.

Afin de réduire la complexité liée au grand nombre de labels, nous utilisons aussi une méthode de composition de champs aléatoires conditionnels. Si les expériences de la section 5.2.5 ont montré que la composition hiérarchique est la solution la plus efficace, elle n'est toutefois pas adaptée à la tâche considérée ici. En effet, bien que l'on dispose d'une DTD de sortie, l'alphabet des labels ne porte que sur les chemins de la racine aux feuilles, c'est-à-dire sur les éléments ayant un contenu textuel. La DTD ne fournit donc pas de hiérarchie sur l'alphabet des labels. Nous utilisons donc la composition séquentielle afin de profiter à la fois de la réduction de la complexité et des dépendances longue distance que cette technique apporte. Pour le choix de la partition de l'alphabet des labels, nous nous sommes aidés de la DTD des arbres XML de sortie. Par exemple, toutes les informations concernant une récompense gagnée par un film (nom, lieu et année) forment une seule sous-partie, tandis que le titre du film, puisqu'il n'est directement lié à aucune autre information, forme une sous-partie à lui tout seul. On obtient ainsi pour le corpus AllMovie, une partition composée de 30 sous-parties de 2 à 6 labels, tandis que l'alphabet des labels pour Allociné et IMDB est divisé en 7 sous-parties, chacune comportant de 2 à 9 labels.

Résultats

Afin d'évaluer les performances des 3-CRFs sur cette tâche, nous mesurons dans un premier temps la qualité de l'annotation des documents en mesurant précision, rappel et F_1-mesure sur l'ensemble des labels de la tâche. Les résultats présentés dans le tableau 6.4 correspondent à la micro moyenne de ces mesures, c'est-à-dire la moyenne tenant compte du nombre d'occurrences de chaque label.

Corpus		Rappel	Précision	F_1-mesure
Allocine		85.58	82.53	83.48
IMDB		88.19	90.34	88.10
AllMovie	html1	87.81	71.41	78.76
	html2	83.25	66.05	73.66

TAB. 6.4 – Qualité de l'annotation sur la tâche de *Structure Mapping* du challenge XML-Mining.

Dans l'ensemble, les 3-CRFs obtiennent de bons résultats en matière d'annotations des arbres XML. Les résultats sont même très bons en annotation pour le corpus IMDB

Corpus		Chemins		Sous-arbres		Chemins + sous-arbres	
Méthode		ISM	3-CRFs	ISM	3-CRFs	ISM	3-CRFs
Allocine		76.80	86.18	65.85	77.99	64.09	77.99
IMDB		84.25	87.80	71.36	78.24	71.18	78.24
AllMovie	html1	-	91.81	-	89.76	-	89.76
	html2	-	79.60	-	71.79	-	71.57

TAB. 6.5 – Évaluation des arbres XML obtenus sur la tâche de Structure Mapping du challenge XMLMining.

avec une F_1-mesure atteignant 88.10. Toutefois, la difficulté des corpus ainsi que le grand nombre de labels, dont certains n'ont que très peu d'occurrences, empêchent d'obtenir des résultats proches des 100%. De plus, au sein du corpus AllMovie, on remarque une nette différence entre les jeux de données html1 et html2. Les résultats sur le jeu de données html2 montrent une nette chute, essentiellement au niveau de la précision. Ce phénomène est dû au grand nombre d'informations inutiles présentes dans les documents XHTML d'entrée de ce jeu de données. Les 3-CRFs ne parviennent alors pas à distinguer correctement les informations qu'il est nécessaire de conserver dans les arbres de sortie de celles qu'il faut supprimer.

Nous évaluons maintenant les performances de notre méthode non plus sur la qualité de l'annotation en elle-même, mais sur la qualité de la transformation. Pour cela, nous utilisons les mesures de comparaisons d'arbres présentées dans la section 2.2.3 mesurant la F_1-mesure sur les chemins, les sous-arbres et les paires (chemin, sous-arbre).

Les résultats avec ces mesures sont présentés dans le tableau 6.5. Celui-ci fournit aussi la comparaison avec les résultats obtenus par l'algorithme ISM [Maes et al., 2007], autre participant de la tâche de Structure Mapping du challenge XML Mining. Pour ce système, aucun résultat n'est fourni concernant le corpus AllMovie, sur lequel l'algorithme ISM ne pouvait être évalué en raison de la nécessité d'identifier plusieurs informations différentes au sein d'une même feuille texte.

En ce qui concerne notre système, les résultats obtenus présentent le même comportement général qu'avec la précédente évaluation. En effet, sur le corpus AllMovie, les résultats sur le jeu de données html1 sont bien meilleurs que ceux obtenus sur html2. De manière plus détaillée, avec html1, on remarque que la F_1-mesure est relativement stable quelle que soit la mesure utilisée. D'une part, le score de près de 92% avec la mesure sur les chemins montre que les bonnes informations textuelles sont identifiées et qu'elles sont correctement intégrées dans l'arbre XML de sortie. D'autre part, le résultat proche de 90% avec les deux autres mesures indique la bonne structure générale des arbres générés. Les résultats sur le corpus html2 sont relativement moins satisfaisants. En effet, si le score avec la mesure sur les chemins avoisine un correct 80%, témoignant d'une bonne identification des portions de texte, les deux autres mesures présentent une baisse de performances significative.

Les résultats obtenus par les champs aléatoires conditionnels sur les corpus Allociné et IMDB sont aussi très bons. En effet, ceux-ci approchent les 80% de F_1-mesure pour les mesures évaluant la structure de l'arbre obtenu, et atteignent même 87% pour la mesure

```
<!ELEMENT rss (channel)>
<!ATTLIST rss version CDATA #FIXED "2.0">
<!ELEMENT channel (item+)>
<!ELEMENT item ((title|description)+,link?,
                (author|category|comments|enclosure|guid|pubDate|source)*)>
<!ELEMENT author (#PCDATA)>
```

FIG. 6.4 – Extrait de la DTD de RSS 2.0.

sur les chemins. De plus, la comparaison avec les résultats de l'algorithme ISM montre la bonne performance de notre système. Toutefois, les résultats présentés pour l'algorithme ISM sont des résultats préliminaires, et l'algorithme présente l'avantage d'être en moyenne 10 fois plus rapide que les champs aléatoires conditionnels pour transformer un arbre XML.

6.2.2 Génération automatique de flux RSS

La seconde tâche de transformation du XHTML vers le XML que nous considérons est une tâche de génération automatique de flux RSS à partir de pages Web correspondant aux résultats d'une recherche sur un site. Cette tâche a donné lieu à la création d'une application en ligne[17] basée sur les champs aléatoires conditionnels et permettant à un utilisateur d'annoter une ou plusieurs pages de résultats d'un site afin d'apprendre à générer automatiquement des flux RSS sur ce site. Nous présentons dans un premier temps cette tâche de transformations d'arbres XML et les performances des champs aléatoires conditionnels avant de présenter plus en détails l'application mise en œuvre.

Tâche et données

Le but de cette tâche est de générer automatiquement des flux RSS personnalisés à partir de pages Web listant les résultats à une recherche sur un site. Un flux RSS est un document XML pour la syndication de contenu Web, utilisé pour diffuser des mises à jour de sites dont le contenu est susceptible de changer régulièrement, comme par exemple les sites d'informations. Un tel document liste un certain nombre d'entrées (`item`), pour lesquelles sont précisées, entre autres, un titre, une description, une date de publication, un auteur, *etc.* Un extrait de la DTD de RSS 2.0 est proposé sur la figure 6.4. Bien que de plus en plus de sites proposent ce type de flux RSS, ce sont pour la plupart des flux prédéfinis et ne répondant donc pas précisément aux critères spécifiques de chaque utilisateur. De plus, de nombreux sites Web ne proposent toujours pas cette fonctionnalité.

L'intérêt d'une telle application de génération automatique de flux RSS est donc qu'elle permet à un utilisateur d'être informé de l'arrivée, sur un site, de nouveaux éléments susceptibles de l'intéresser, car répondant aux critères d'une recherche personnalisée. Par exemple, si un utilisateur est intéressé, sur un site de photographie, par les photos à dominante de couleur rouge, cette application propose de générer automatiquement un flux RSS à partir de la page de résultats à la recherche "red OR rouge" qui retourne toutes

[17] `http://cluster-mostrare.futurs.inria.fr:8080/xcrf2rss-project`

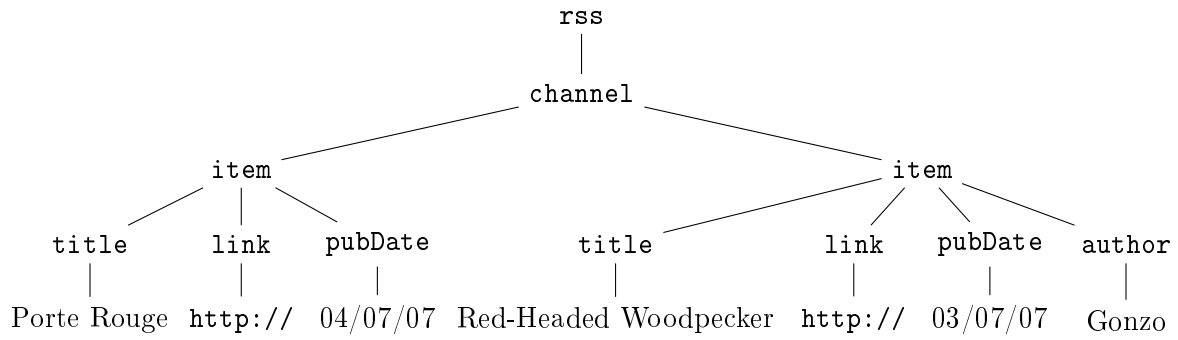

FIG. 6.5 – Exemple de flux RSS.

les photos ayant un de ces deux mots-clés. Ce flux RSS, mis à jour régulièrement, permet ainsi à l'utilisateur d'être informé de toute nouvelle photo répondant à cette requête. Cette fonctionnalité est déjà proposée par certains sites tels que DailyMotion ou Google Video, mais reste toutefois très marginale. Dans le cas de cette application, les flux RSS sont alors des documents XML listant les résultats de la recherche effectuée (des photos dans le cas de l'exemple précédent). Pour chaque entrée, des informations telles que le titre, la description, le lien vers la page correspondant à cette entrée sont répertoriées. La figure 6.5 présente un exemple d'arbre XML de flux RSS. Ce flux présente deux résultats de la recherche "red OR rouge" sur un site de photographie dont les titres sont respectivement "Porte Rouge" et "Red-Headed Woodpecker". Les deux entrées ont un titre, un lien et une date de publication. La seconde entrée répertorie aussi le nom de la personne ayant pris la photo en question.

Pour évaluer les performances des champs aléatoires conditionnels sur cette tâche, nous avons constitué trois corpus à partir de sites connus. Pour cela, nous avons donc téléchargé des pages de résultats de recherches sur les sites Slashdot[18] (informations), Google Video[19] (vidéo en ligne) et Flickr[20] (photographie). Chacun des trois corpus ainsi constitué comporte environ 30 documents XHTML de très grande taille. En effet, le nombre moyen de nœuds (éléments, attributs et nœuds texte) est d'environ 6000. Ces arbres XHTML sont par définition orientés documents. Toutefois, les pages de ces sites étant générées automatiquement, les nœuds XHTML possèdent souvent des attributs offrant une information plus riche qu'une simple information de présentation. Un cas extrême de ce comportement est visible dans le corpus constitué de page issues de Google Video. En effet, le titre d'un résultat de la recherche est contenu dans une balise `div` possédant un attribut `class` dont la valeur est `title`. Un exemple simplifié d'arbre HTML extrait du corpus Slashdot est représenté en figure 6.6.

Modélisation de la transformation

Du fait du format XML des flux RSS, la tâche de génération automatique de flux RSS est donc bien une tâche de transformation du XHTML vers le XML. Nous pouvons donc

[18] `http://www.slashdot.org`
[19] `http://video.google.com`
[20] `http://www.flickr.com`

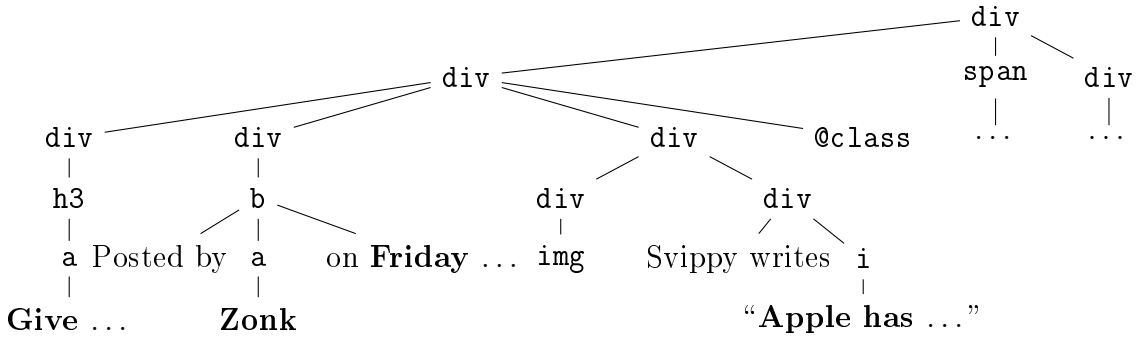

FIG. 6.6 – Exemple simplifié d'arbre XHTML du corpus Slashdot.

l'aborder comme une tâche d'annotation à l'aide des champs aléatoires conditionnels. Pour cela, nous utilisons la méthode d'annotation avec des opérations d'édition décrite dans la section 1.4.2.

La DTD de RSS 2.0, le format de sortie de cette tâche, possède 30 éléments différents. Si l'on considère toutes les opérations d'édition possibles (renommer, insérer, supprimer un nœud, supprimer un sous-arbre et ne rien faire), la taille de l'alphabet des labels pour cette tâche est donc théoriquement de 63 labels. Toutefois, de nombreux éléments définis dans cette DTD correspondent soit à des informations qui ne sont pas nécessaires, soit à des informations qui ne sont pas présentes dans les pages de résultats des sites. Par exemple, une page de résultats sur le site Flickr ne propose pas de description des photos. Ainsi, nous avons adapté à chaque corpus l'alphabet des labels correspondant. La taille des alphabets oscille entre 12 et 16 labels.

Choix du modèle

Nous utilisons, une fois de plus, le modèle des 3-CRFs pour effectuer cette tâche d'annotation, toujours dans le but d'exploiter au maximum les dépendances possibles entre labels.

La définition de cette tâche permet aussi d'intégrer de nombreuses contraintes dans les champs aléatoires conditionnels. En effet, même si l'alphabet des labels est constitué d'opérations d'édition et non des simples éléments de la DTD de RSS 2.0, l'annotation effectuée doit être cohérente avec la DTD. Ainsi, si l'on considère que dans la DTD de RSS 2.0, un élément `title` ne peut contenir que du texte, il en résulte que dans l'arbre XHTML annoté, il ne peut y avoir, sous un nœud annoté par `title` ou `insert_title` que des suppressions de nœuds éléments et les nœuds texte sont inchangés. De plus, un élément `item` de la DTD RSS 2.0 doit obligatoirement contenir au moins un titre et par conséquent posséder un fils. Un nœud de l'arbre XHTML annoté par `item` ne peut donc pas être un nœud texte, qui par définition n'a pas de fils.

Nous n'utilisons par contre pas de méthode de composition. En effet, la taille de l'alphabet des labels varie, selon le corpus, entre 12 et 16 labels, ce qui est raisonnable du point de vue de la complexité des algorithmes d'inférence et d'apprentissage.

Source	Précision	Rappel	F-mesure
Google Video	96.54	98.82	97.49
Slashdot	99.66	97.28	98.43
Flickr	100	100	100

TAB. 6.6 – Qualité de l'annotation pour la tâche de génération de flux RSS.

Source	Chemins	Sous-arbre	Chemin et Sous-arbres
Google Video	100	100	100
Slashdot	100	100	100
Flickr	100	100	100

TAB. 6.7 – Évaluation des arbres RSS générés.

Résultats

Nous évaluons les 3-CRFs avec les contraintes définies précédemment selon deux critères : la qualité de l'annotation et la similarité des arbres XML générés avec ceux attendus.

Le tableau 6.6 présente les résultats en annotation des 3-CRFs sur les trois corpus considérés pour cette tâche. Nous avons choisi de présenter, pour chaque corpus, la macro moyenne des F_1-mesures obtenues à chaque label. Le choix de cette moyenne a été guidé par le fait que, dans cette tâche, les nœuds sont pour la plupart (au moins 90%) annotés par les labels `delete` et `delST`. La forte représentation de ces labels aurait donc entraîné un biais avec la micro moyenne.

On observe que les champs aléatoires obtiennent des résultats proches des 100% en termes d'annotation sur les 4 jeux de données sur lesquels nous les avons testés, aussi bien en terme de précision que de rappel. Le score maximal est même atteint pour le jeu de données issues de Flickr, ce qui n'est pas étonnant. En effet, non seulement ce corpus est très fortement structuré, mais il contient aussi de nombreuses informations dans les attributs. Par exemple, dans les documents XHTML issus de Flickr, chaque entrée (`item`) du flux RSS est contenue dans tableau, celui-ci ayant un attribut `class` dont la valeur est "DetailResult". Ceci permet aux 3-CRFs d'identifier très facilement chaque entrée du flux à générer.

Nous mesurons maintenant la qualité des flux RSS générés par la transformation effectuée à l'aide des annotations des 3-CRFs. Les résultats selon les trois critères de similarité d'arbres (décrits dans la section 6.2.1.0) sont présentés dans le tableau 6.7. On remarque que les flux RSS générés sont en tous points identiques à ceux attendus. Ces résultats peuvent être surprenants, étant donné que l'évaluation de l'annotation ne donnait pas des résultats parfaits. Toutefois ceci s'explique par le fait que, comme expliqué dans la section 1.4.2, plusieurs annotations différentes peuvent aboutir à un même arbre XML de sortie. Ainsi, les quelques erreurs d'annotations constatées précédemment portaient uniquement des labels de suppression de nœuds (`delete` et `delST`), remplaçant ainsi la suppression complète d'un sous-arbre par la suppression individuelle de chacun des nœuds de ce sous-arbre. Ces deux annotations ayant le même résultat lorsque la transformation est effectuée, les flux RSS générés ne sont pas compromis. La qualité de ces résultats ainsi

que la possible portée de cette tâche nous ont conduit à réaliser une application en ligne de génération de flux RSS basée sur les champs aléatoires conditionnels.

6.3 Outil en ligne de génération de flux RSS

Dans le cadre de cette thèse, nous avons donc développé un outil en ligne de génération automatique de flux RSS personnalisés à partir de pages Web. Le principe général de cet outil est le suivant : l'utilisateur annote manuellement une page Web issue du site sur lequel il veut générer un flux RSS. Cette page est utilisée comme exemple pour l'apprentissage d'un champ aléatoire conditionnel pour les arbres XML. Celui-ci est ensuite utilisé pour permettre à l'utilisateur de générer automatiquement des flux RSS personnalisés sur ce site. Afin de garantir une compatibilité maximale multi-plateformes, cet outil a été développé sous la forme d'un site Web 2.0 en AJAX au moyen de la boîte à outils GWT[21] (Google Web Toolkit) fournie par Google.

6.3.1 Exemple d'utilisation

Nous allons maintenant décrire le fonctionnement de cette application sur un exemple. Supposons qu'un utilisateur veuille générer, à partir du site Google Scholar[22] un flux RSS concernant les articles traitant des champs aléatoires conditionnels, c'est-à-dire les articles répondant à la requête "conditional random fields". Dans un premier temps, afin d'apprendre un champ aléatoire conditionnel pour le site, l'utilisateur doit donc annoter manuellement une page de résultats issue de ce site, cette page n'étant pas nécessairement celle sur laquelle le flux sera généré. Nous choisissons d'annoter la page de résultats à la recherche "machine learning" pour cette première phase. L'application fournit alors une interface d'annotation comme le montre la figure 6.7. Cette interface propose, dans sa partie supérieure, un menu permettant de choisir le type d'élément à annoter (titre, description, lien, auteur ou image). Au centre de l'interface se trouve la page à annoter, en l'occurrence la page issue de Google Scholar listant les résultats de la recherche "Machine Learning". L'annotation se fait en sélectionnant le type (par exemple "title") dans le menu en haut, puis en cliquant sur la zone de texte à annoter. Comme on peut le voir sur le quatrième résultat de la page, par souci de clarté, la zone de texte sur laquelle porte l'annotation est encadrée lors du passage de la souris. Lorsqu'une zone est annotée, celle-ci est surlignée d'une couleur correspondant à son type. Sur l'exemple, les zones en rouge correspondent aux titres des entrées, tandis que les zones en vert correspondent aux annotations. Pour annuler l'annotation d'une zone de texte, il suffit de cliquer à nouveau dessus. Une fois l'annotation terminée, l'utilisateur a le choix entre annoter une seconde page afin de parfaire l'apprentissage dans le cas de sites que l'on qualifiera de "complexes", ou il peut lancer l'apprentissage du champ aléatoire conditionnel en cliquant sur le bouton "Save". L'application propose alors de saisir une description du CRF qui sera appris sur cette page. Celui-ci, une fois l'apprentissage terminé, est stocké dans la liste des générateurs de flux RSS de l'utilisateur.

[21] `http://code.google.com/webtoolkit`
[22] `http://scholar.google.com`

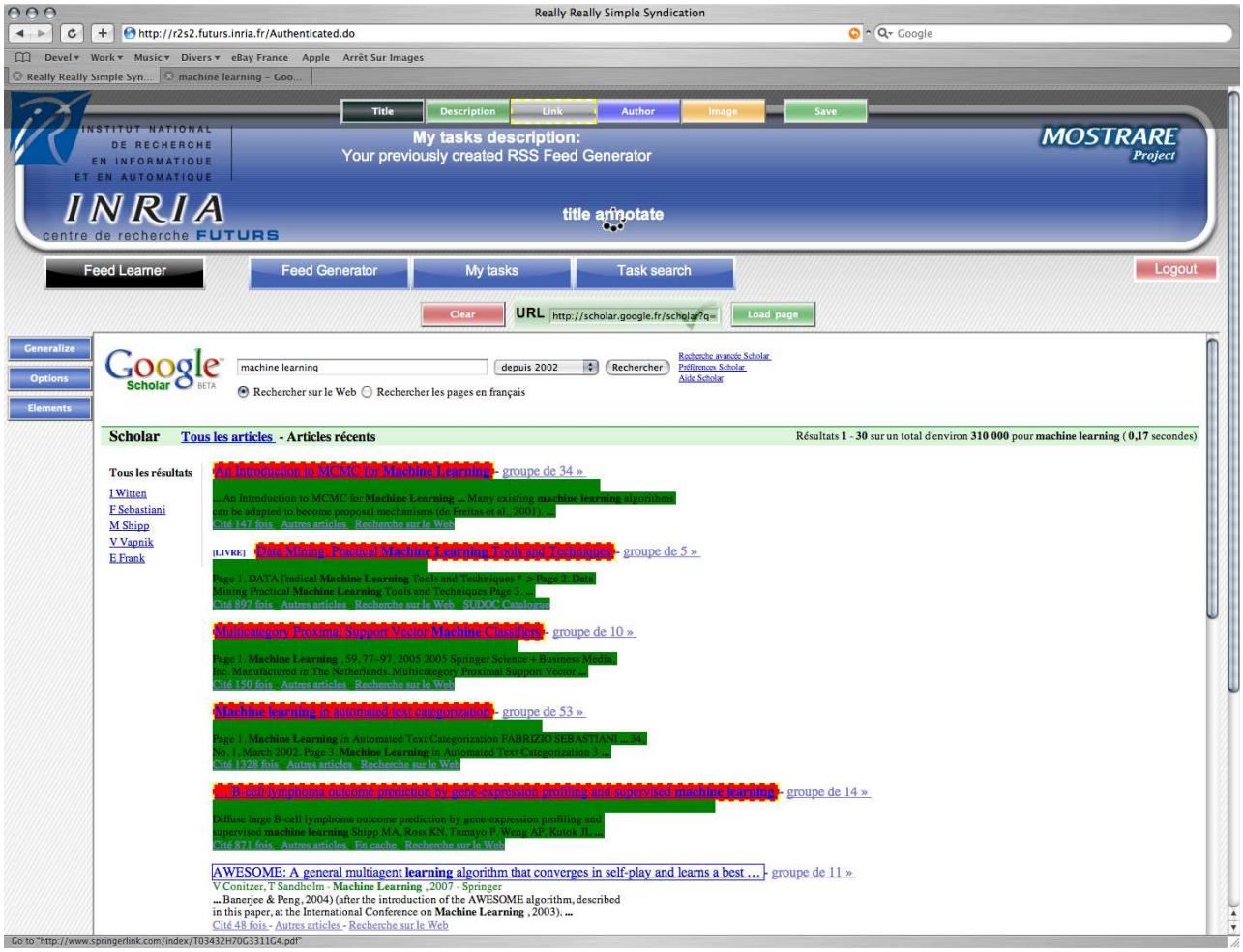

FIG. 6.7 – Interface d'annotation de l'application RSS.

À l'aide du champ aléatoire conditionnel ainsi appris, l'utilisateur peut alors générer automatiquement le flux RSS qui l'intéresse sur ce site. Pour cela, il lui suffit de sélectionner le CRF et de saisir, dans l'interface, l'adresse de la page sur laquelle il désire générer un flux. Dans le cas de notre exemple, cette page est la page de résultats à la recherche "conditional random fields". Il saisit aussi le titre qu'il désire attribuer au flux RSS : dans notre exemple, ce titre est "googleCRF". L'application génère alors le flux, comme le montre la figure 6.8. Celui-ci peut être ouvert avec n'importe quel lecteur de flux RSS.

Durant le développement de cette application, nous avons découvert l'existence d'une autre plateforme proposant ce même service de génération automatique de flux RSS à partir de pages Web appelée "Dapper : The Data Mapper"[23]. Si cette application offre une interface utilisateur plus avancée et permet non seulement de transformer des données au format RSS, mais aussi vers d'autres formats XML, tels que celui utilisé par Google Maps, elle ne bénéficie pas de la qualité des champs aléatoires conditionnels en terme d'apprentissage. Ainsi, sur des sites possédant une structure complexe, ou à l'inverse ayant une structure très légère (l'essentiel du contenu étant dans les feuilles) Dapper a tendance

[23] `http://www.dapper.net`

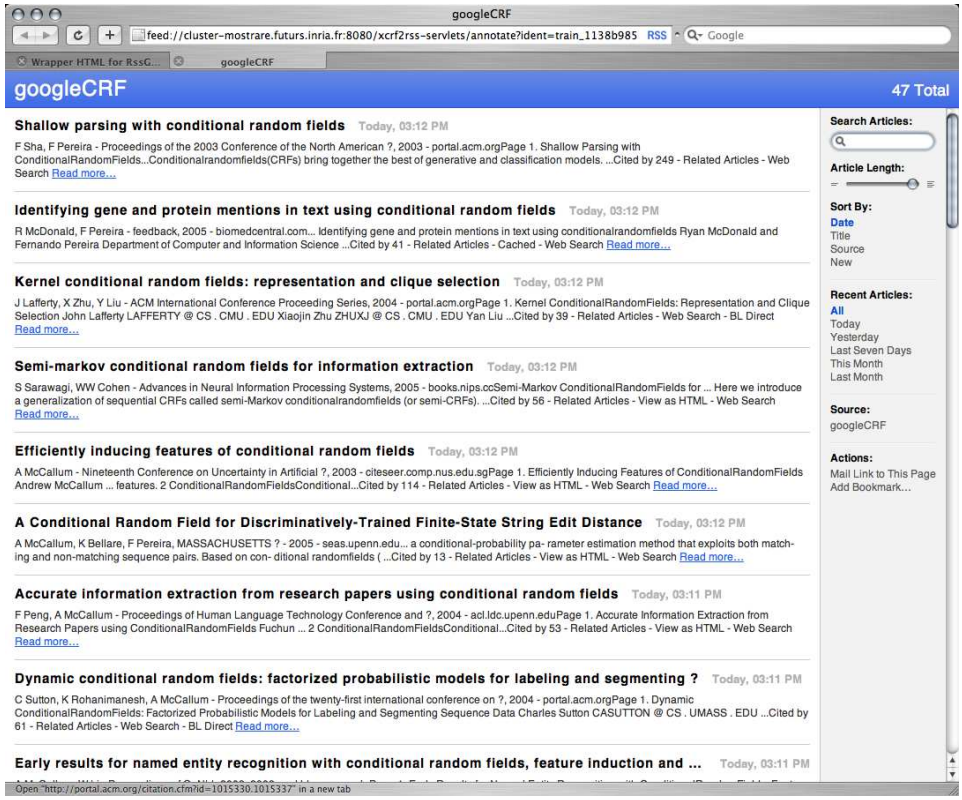

FIG. 6.8 – Flux RSS pour "conditional random fields" généré par l'application RSS.

à surgénéraliser et extraire des informations inutiles, tandis que notre application génère un flux RSS correct.

6.3.2 Fonctionnement de l'outil

Pour cet outil, nous avons choisi de modéliser, comme dans l'expérience précédente (*cf.* section 6.2.2), les transformations d'arbres XHTML en arbres XML au format RSS par des annotations de type "opérations d'édition" des arbres XHTML d'entrée. Dans un souci de rapidité d'exécution, nous avons réduit au maximum la taille de l'alphabet des labels en ne considérant que les éléments importants de la DTD de RSS que sont `channel`, `item`, `title`, `author`, `description`, `pubDate` et `enclosure`.

Le modèle de dépendances que nous avons choisi est celui des 3-CRFs, celui-ci permettant de modéliser un maximum de dépendances entre les labels, offrant ainsi les meilleurs résultats. Toutefois, afin de préserver la rapidité d'exécution, nous avons intégré les contraintes provenant à la fois de la nature de l'alphabet des labels (labels de type "opération d'édition") et de la DTD de RSS. Ces contraintes sont décrites plus avant dans la section 6.2.2.

De plus, afin de garantir la simplicité d'utilisation de cette application, il n'était pas envisageable de demander à l'utilisateur, lors de la phase d'annotation manuelle, d'annoter les nœuds internes des documents avec des labels du type `delete`, ou `item`. Ainsi, l'utilisateur ne doit annoter que les éléments correspondant aux feuilles de la DTD de

RSS. Une fois cette annotation effectuée, une feuille de transformation XSLT est appliquée. Celle-ci se charge de terminer l'annotation. Plusieurs cas sont à considérer :

- il faut dans un premier temps identifier les nœuds à supprimer. Ceux-ci se divisent entre les nœuds n'ayant aucun fils annoté et ceux en ayant. Les premiers sont annotés par `delST`, c'est-à-dire supprimer le sous-arbre, tandis que les autres sont annotés par `delete`.
- il faut aussi retrouver quel nœud correspond à un élément d'information, c'est-à-dire quel nœud doit être annoté par `item`. Nous utilisons pour cela la connaissance de la DTD de RSS. En effet, un élément `item` ne peut contenir qu'une seule occurrence de chacun de ses éléments fils (`title`, `description`, *etc.*). Ainsi, le nœud annoté par `item` est le nœud ayant dans son sous-arbre au plus un nœud annoté par un label parmi {`title`,`description`,...} et dont le père possède deux nœuds annotés par un même label dans cet ensemble. Cette expression fait l'hypothèse que la page Web annotée contient au moins deux informations. Dans le cas contraire, la condition sur le père est tout simplement qu'il soit la racine de l'arbre.
- Enfin, il faut aussi annoter par `channel` le nœud correspond à l'ensemble du flux RSS. Nous avons choisi par souci de simplicité de toujours annoter la racine de l'arbre XHTML avec ce label.

Cette feuille de transformation est donc utilisée avant de lancer l'apprentissage du 3-CRF.

Lors de la génération d'un flux RSS à l'aide d'un CRF déjà appris, celui-ci annote le document XHTML choisi par l'utilisateur. Une simple feuille de transformation XSLT vient alors transformer ce document en arbre XML au format RSS en suivant les instructions des labels. Cet arbre peut alors être transmis à l'utilisateur.

6.4 Conclusion sur les expériences

À travers les différentes expériences de ce chapitre, nous avons montré l'efficacité de notre méthode de transformation d'arbres XML. En effet, dans un premier temps, l'utilisation des 3-CRFs et des méthodes d'annotation pour la transformation d'arbres XML que nous avons proposées a permis d'obtenir, dans chacune des tâches présentées, de très bons résultats. De plus, le type de documents XML en entrée, c'est-à-dire les arbres XML orientés données ou ceux orientés document comme les arbres XHTML, ne change pas la qualité des résultats. Mais ces expériences montrent aussi que, à travers l'utilisation de contraintes ou de méthodes de composition de champs aléatoires conditionnels, ces 3-CRFs peuvent être utilisés dans le cadre d'expériences à grande échelle. En effet, que ce soit sur des arbres de grande taille comme c'est le cas dans l'expérience de génération automatique de flux RSS, ou sur des corpus composés de plusieurs milliers de documents comme dans la tâche de *Structure Mapping* du challenge XML Mining 2006, les 3-CRFs ont été utilisés avec succès. Enfin, le succès de notre méthode dans les expériences de génération automatique de flux RSS nous a conduit à réaliser un prototype d'application très performant dans ce domaine.

Conclusion

1 Bilan

Nous avons montré dans cette thèse qu'il est possible d'effectuer des tâches de transformation d'arbres XML à l'aide de modèles probabilistes pour l'annotation. Pour cela nous avons :

- proposé deux méthodes d'annotation d'arbres XML définissant des transformations de ces arbres ;
- adapté les champs aléatoires conditionnels au cas de l'annotation d'arbres XML ;
- défini des contraintes pour les champs aléatoires conditionnels permettent de réduire la complexité des algorithmes d'inférence exacte et d'apprentissage ;
- défini des méthodes de composition de CRFs permettant d'approximer un CRF par plusieurs CRFs de complexités inférieures.

Dans un premier temps, nous avons donc proposé deux méthodes d'annotation des nœuds d'arbres XML permettant de définir des transformations de ces arbres. La première, qui consiste à annoter les feuilles des arbres d'entrée avec leur chemin dans l'arbre de sortie, possède l'intérêt de permettre de représenter des transformations d'arbres XML complexes où les structures des arbres d'entrée et de sortie sont très différentes. Toutefois, l'ordre des feuilles dans l'arbre de sortie est largement conditionné par l'arbre d'entrée. De plus, en n'annotant pas les nœuds internes, cette méthode ne permet pas d'exploiter au mieux les dépendances entre labels que permettent de modéliser les CRFs. La seconde méthode consiste, quant à elle, à annoter les nœuds (éléments, attributs et textes) des arbres XML d'entrée avec des opérations d'édition d'arbres. Contrairement à la méthode précédente, celle-ci permet de tire au maximum parti des dépendances entre les labels des nœuds internes. Toutefois, elle limite les transformations à des cas plus simples dans lesquels le squelette de l'arbre de sortie est inclus, à quelques insertions de nœuds près, dans celui de l'arbre d'entrée.

Pour utiliser ces deux méthodes de transformation par l'annotation, nous avons ensuite adapté le modèle des champs aléatoires conditionnels, essentiellement utilisé pour l'annotation de séquences, au cas des arbres XML. Pour cela, nous avons proposé trois modèles de dépendances de complexité croissante. Parmi ceux-ci, le modèle des 3-CRFs utilise les deux dimensions horizontale et verticale des arbres XML pour modéliser des dépendances. Nous avons aussi adapté les algorithmes d'inférence exacte et d'apprentissage qui lui sont associés. Un série d'expériences sur des tâches d'annotation d'arbres XML a montré à la fois la qualité de ce modèle et ses limites. En effet, d'une part, les 3-CRFs per-

mettent d'obtenir d'excellents résultats, tout en ne nécessitant que très peu d'exemples en apprentissage et une connaissance limitée sur les arbres XML d'entrée. Malheureusement, la complexité des algorithmes d'inférence exacte et d'apprentissage pour les 3-CRFs est cubique en la taille de l'alphabet des labels. Cela rend par conséquent l'utilisation de ce modèle impossible dans le cadre d'application en ligne ou à grande échelle, c'est-à-dire avec un alphabet des labels de grande taille.

Pour remédier à ce problème et permettre ainsi d'utiliser les CRFs, et plus particulièrement les 3-CRFs, dans des applications à grande échelle, nous avons donc proposé deux méthodes d'amélioration de la complexité des CRFs. Celles-ci, contrairement aux algorithmes existant d'inférence approchée, exploitent de deux façons différentes les connaissances du domaine pour réduire la complexité. Dans un premier temps, nous avons défini des contraintes pour les champs aléatoires conditionnels. Celles-ci permettent de réduire le nombre d'annotations possibles d'un arbre d'entrée en interdisant des configurations de labels au niveau des cliques du graphe d'indépendances. La complexité des algorithmes d'inférence exacte et d'apprentissage pour les CRFs est ainsi réduite ainsi. De plus, ces contraintes étant écrites à partir de connaissances du domaine (par exemple, le schéma des arbres XML de sortie), elles permettent aussi d'éliminer des annotations incohérentes et de cette façon peuvent améliorer les résultats, particulièrement avec peu d'exemples en apprentissage. Toutefois, l'amélioration de complexité occasionnée par ces contraintes reste légèrement insuffisante pour une utilisation des 3-CRFs dans des applications *online*.

La seconde méthode d'amélioration de la complexité que nous avons introduite est la définition de trois méthodes de composition de champs aléatoires conditionnels. Celles-ci permettent d'approximer un CRF défini sur un grand alphabet de labels par plusieurs CRFs définis sur des sous-parties de cet alphabet. Pour trouver ces sous-parties, ces méthodes de composition s'appuient sur l'existence d'une partition de l'alphabet des labels ou d'une hiérarchie sur ces labels. La complexité des algorithmes d'inférence et d'apprentissage des 3-CRFs étant cubique en la taille de l'alphabet des labels, cette méthode permet, selon la taille des sous-parties de l'alphabet des labels, de réduire considérablement la complexité de ces algorithmes. Ces méthodes nous permettent alors d'utiliser les 3-CRFs dans le cadre d'applications à grande échelle, quelle que soit la taille de l'alphabet des labels. Toutefois, elles nécessitent pour cela la connaissance soit d'une partition naturelle de cet alphabet, soit d'une hiérarchie sur les labels. Si, dans le cas de la transformation d'arbres XML, le schéma de sortie des arbres XML offre une telle connaissance, d'autres applications ne permettent pas toujours de définir de telles partitions ou hiérarchies.

Enfin, sur le plan pratique, nous avons appliqué les 3-CRFs à diverses tâches de transformations d'arbres XML. Par le biais de ces expériences, nous avons montré la pertinence de notre approche pour la transformation d'arbres XML, notamment en la comparant à des systèmes de *Schema Matching* et de transformation. Ces expériences nous ont amené à réaliser une application Web de génération automatique de flux RSS. La mise en œuvre et les bonnes performances de cette application RSS sont une preuve de la qualité de nos travaux dans le domaine. Elle prouve aussi, aux côtés d'autres programmes tels que *Squirrel* [Carme, 2005], que l'apprentissage automatique permet d'obtenir d'excellents résultats dans le cadre d'applications ayant trait aux données du Web.

2 Perspectives

Si ces résultats appuient effectivement notre thèse et montrent que l'utilisation de champs aléatoires conditionnels pour les arbres XML permet d'aborder des tâches de transformation d'arbres, nous pensons que ces travaux en appellent aussi de nombreux autres.

Dans un premier temps, nous pensons que le modèle des 3-CRFs peut être utilisé dans le cadre d'autres tâches que la transformation d'arbres. En effet, les CRFs sur les séquences ont été, dans de nombreux travaux, appliqués avec succès dans le cadre du traitement des langues naturelles, comme par exemple l'annotation *Part-Of-Speech*. De plus, dans ce domaine, plusieurs corpus arborés sont disponibles, comme le corpus d'Anne Abeillé [Abeillé et al., 2003] composé de phrases en français extraites du journal *Le Monde* ou, pour la langue anglaise, le *Penn Treebank* [Marcus et al., 1993]. Dans ces deux corpus, la structure arborescente correspond aux arbres d'analyse syntaxique des phrases. Une tâche comme l'annotation des rôles sémantiques peut être effectuée sur ces corpus arborés. Celle-ci étant une tâche d'annotation d'arbres, nous pensons que les 3-CRFs sont naturellement adaptés et peuvent donc obtenir de bons résultats. Des expériences sont en cours pour cette tâche sur le corpus d'Anne Abeillé et pour la tâche de *Semantic Role Labeling* de CONLL'05 [Carreras and Marquez, 2005] (sur le *Penn Treebank*). Cette dernière expérience permettra de plus de comparer les performances des 3-CRFs aux 19 autres systèmes qui y ont participé en 2005.

Une autre extension naturelle de cette thèse concerne les techniques d'apprentissage pour les 3-CRFs. Dans ce sens, des travaux sur l'apprentissage non supervisé de 3-CRFs sont en cours. Ceux-ci se font dans le cadre de l'annotation de pages Web listant des références bibliographiques, l'objectif étant d'identifier dans ces pages les différents articles et leurs détails : titre, liste des auteurs, date de publication, journal, *etc.* Afin d'effectuer de l'apprentissage non supervisé, ces travaux consistent donc à apprendre des 3-CRFs à partir d'arbres XHTML non annotés. Dans un premier temps, ces arbres sont donc annotés à l'aide de dictionnaires (*gazetteers*) de titres, nom d'auteurs ou de conférences, ou encore à l'aide d'expressions régulières reconnaissant les dates. Ces premières annotations ne sont que partiellement correctes : certaines informations sont erronées tandis que d'autres n'ont pas été trouvées. Toutefois, un 3-CRF est appris sur ces annotations. L'hypothèse faite ici est que les 3-CRFs, en utilisant les régularités de structure de ces documents XHTML, viendront corriger ces erreurs et ainsi obtenir de meilleurs résultats. De premières expériences, effectuées par Daniel Muschick dans le cadre d'un stage de Master 2, ont validé cette hypothèse et nous encouragent donc à poursuivre les travaux dans cette voie.

Dans un futur proche, plusieurs autres extensions de nos travaux sont aussi envisageables. Tout d'abord, nous estimons qu'il peut être intéressant d'améliorer dans les 3-CRFs l'utilisation conjointe des vues arborescentes et textuelles des arbres XML, et plus particulièrement dans les documents XHTML. En effet, dans ces arbres, la séquence des feuilles texte forme un ensemble cohérent correspondant au texte affiché par le navigateur. Dans le modèle des 3-CRFs, par l'intermédiaire des fonctions de caractéristiques, il est possible de prendre en compte le contenu textuel des arbres XHTML avec des tests sur l'observation. Toutefois, avec ce modèle de dépendances, deux feuilles texte consécutives mais ayant deux pères différents sont considérées indépendantes. Ainsi, le label

affecté à l'une ne peut influer sur le choix du label de l'autre. Pourtant, nous pensons que l'utilisation de ce type de dépendances pourrait aider à améliorer encore la qualité de l'annotation. Une piste pour modéliser ces dépendances est alors de combiner les CRFs sur les séquences et les 3-CRFs.

Enfin, le développement de l'application de génération automatique de flux RSS nous amène à réfléchir sur la possibilité d'apprendre des 3-CRFs à partir de documents partiellement annotés, c'est-à-dire d'arbres XML dans lesquels le label associé à certains nœuds n'est pas déterminé. En effet, l'annotation complète d'une page Web est une tâche fastidieuse pour l'utilisateur. De plus, de par la régularité de ces pages Web générées automatiquement par des programmes, l'annotation d'une ou deux entrées (`item`) dans ces pages devrait être, dans la plupart des cas, suffisante pour apprendre un 3-CRF capable d'annoter correctement les arbres. Pour permettre cela, une piste possible consiste à s'inspirer des techniques d'élagage des arbres d'entrée et d'apprentissage interactif proposées dans [Carme et al., 2007]. Une autre piste serait d'utiliser des champs aléatoires conditionnels avec des variables cachées et des algorithmes d'apprentissage de type EM (*Expectation-Maximization*).

Annexe A

Génération automatique des fonctions de caractéristiques

Nous présentons dans cette annexe la procédure de génération automatique des fonctions de caractéristiques. La réalisation d'une telle procédure s'est avérée nécessaire pour de multiples raisons. Dans un premier temps, afin d'obtenir dans nos expériences des résultats montrant l'intérêt effectif du modèle et non l'intérêt d'un ensemble de fonctions de caractéristiques particulièrement adapté à une tâche donnée, nous avons mis en place cette procédure de la façon la plus indépendante du domaine possible. L'ensemble de fonctions de caractéristiques généré contient des tests génériques sur la structure des arbres ainsi que sur leur contenu textuel. L'autre raison qui a nécessité la mise en œuvre de cette procédure de génération concerne les méthodes de composition de champs aléatoires conditionnels. En effet, pour que leur fonctionnement puisse être automatisé, ces méthodes nécessitent une fonction de génération de fonctions de caractéristiques. Nous décrivons cette procédure en détails dans le cas des champs aléatoires conditionnels pour l'annotation d'arbres XML.

La procédure de génération automatique de fonctions de caractéristiques est une fonction Gen qui prend en paramètres un échantillon d'apprentissage $S = \{(\mathbf{x}^j, \mathbf{y}^j)\}_{j=1}^{j=m}$ composé de couples (observation, annotation) ainsi qu'un nombre compris entre 0 et 1 qui sert de seuil de sélection des fonctions de caractéristiques générées et que nous décrirons plus tard. En sortie, cette procédure fournit un ensemble de fonctions de caractéristiques booléennes, qui retournent vrai si le(s) test(s) sur l'annotation et un éventuel test sur l'observation sont satisfaits.

A.1 Pré-traitement

Avant de générer automatiquement les fonctions de caractéristiques à partir de l'échantillon d'apprentissage, une phase de pré-traitement vient calculer de nouveaux attributs sur les arbres XML de l'échantillon. Ces attributs sont purement informatifs et ne constituent pas des données à annoter : ils ne sont donc pris en compte ni lors de l'apprentissage des champs aléatoires conditionnels, ni lors de l'annotation. Ces attributs pré-calculés apportent des informations supplémentaires sur la structure des arbres XML ainsi que sur

leur contenu textuel. Les tableaux A.1 et A.2 proposent la liste exhaustive des attributs que nous avons choisi de pré-calculer. Naturellement, de nombreux autres attributs pourraient apporter des informations pertinentes, mais nous nous sommes limités à ceux-ci, qui permettent déjà d'obtenir d'excellents résultats.

Attribut	Description
nbChildren	nombre de fils du nœud
depth	profondeur du nœud
childPos	position du nœud dans l'ordre des fils de son père

TAB. A.1 – Attributs de structure calculés lors du pré-traitement.

Attribut	Description
containsComma	le texte contient une virgule
containsColon	le texte contient un " :"
containsSemiColon	le texte contient un " ;"
containsAmpersand	le texte contient une esperluette
containsArobas	le texte contient une arobe
isUpperCase	le texte est en majuscules
firstUpperCase	le premier caractère du texte est en majuscule
onlyDigits	le texte ne contient que des chiffres
oneDigit	le texte est un chiffre seul
containsDigits	le texte contient des chiffres
length	longueur du texte
rowHeader	valeur textuelle de l'en-tête de ligne dans le tableau (seulement en HTML)
columnHeader	valeur textuelle de l'en-tête de colonne dans le tableau (seulement en HTML)

TAB. A.2 – Attributs textuels calculés lors du pré-traitement.

A.2 Génération des fonctions

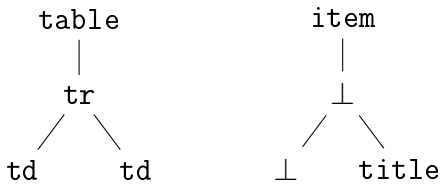

FIG. A.1 – Exemple de couple (observation,annotation) pour la génération de fonctions de caractéristiques.

Une fois la phase de pré-traitement effectuée, l'algorithme 8 de génération automatique des fonctions de caractéristiques peut être appliqué. Cet algorithme, qui prend donc

en entrée un échantillon d'apprentissage S et un seuil σ, commence par initialiser l'ensemble $\mathcal{F}$ des fonctions de caractéristiques. Ensuite, l'algorithme parcourt itérativement chaque couple exemple $(\mathbf{x}^j, \mathbf{y}^j)$ de l'échantillon d'apprentissage. Pour chaque exemple, l'algorithme parcourt alors l'ensemble $\mathcal{C}(\mathbf{y}^j)$ des cliques de l'arbre d'annotation $\mathbf{y}^j$. Pour chaque clique $c \in \mathcal{C}(\mathbf{y}^j)$ est alors générée une fonction de caractéristiques appelée **fonction de caractéristiques structurelle** qui n'effectue aucun test sur l'observation mais qui teste l'agencement des labels y_c dans la clique c considérée. Par exemple, sur le couple (simplifié) de la figure A.1, les fonctions de caractéristiques suivantes sont générées :

$$f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = \texttt{item}, y_{ni} = \perp \\ 0 & \text{sinon} \end{cases}$$

$$f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = \perp, y_{ni} = \perp, y_{n(i+1)} = \texttt{title} \\ 0 & \text{sinon} \end{cases}$$

Il n'est naturellement pas utile de générer de telles fonctions de caractéristiques pour les cliques composées d'un unique nœud.

Ensuite, toujours pour une clique c donnée identifiée par la position n, des fonctions de caractéristiques composées d'un test sur les labels y_c et d'un test sur l'observation $\mathbf{x}$ sont générées pour chaque test sur l'observation possible. Les tests sur l'observation $\mathbf{x}$ sont de la forme $P(x_k)$, où x_k appartient à l'ensemble $N_v(x_n)$ des ancêtres et frères du nœud x_n qui sont à une distance inférieure à v, et où P est un prédicat. L'ensemble $N_v(n)$ est aussi appelé **voisinage** de x_n. On remarquera que nous ne considérons pas dans le voisinage les descendants de x_n. En effet, les arbres XML étant par définition des arbres d'arité non bornée, le nombre de tests deviendrait inutilement grand. En variant la valeur du paramètre v, aussi appelé **paramètre de voisinage**, on fait varier la quantité d'informations sur l'observation dont dispose le champ aléatoire conditionnel.

Les prédicats P sur l'observation peuvent être de deux types différents. On a d'une part des tests sur les symboles de l'observation $\mathbf{x}$ du type $\mathbf{x}_n = a$, où $a \in \mathcal{X}$. Avec ce type de test, on génère sur l'exemple de la figure A.1 des fonctions de caractéristiques de la forme :

$$f_k^{(1)}(y_n, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = \texttt{item}, x_n = \texttt{table} \\ 0 & \text{sinon} \end{cases}$$

$$f_k^{(3)}(y_n, y_{ni}, y_{n(i+1)}, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = 0, y_{ni} = 0, y_{n(i+1)} = \texttt{title} \\ & \quad \text{et le père de } x_n \text{ est } \texttt{table} \\ 0 & \text{sinon} \end{cases}$$

D'autre part, on dispose aussi de tests sur les attributs des nœuds de l'observation. Ces tests concernent non seulement les attributs présents dans les documents XML d'origine, par exemple l'attribut `href` d'un lien HTML, mais aussi les attributs calculés lors du prétraitement des données. Par exemple, sur le couple de la figure A.1, les fonctions suivantes peuvent être générées :

$$f_k^{(1)}(y_n, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = \texttt{title}, \text{ et } x_n \text{ a l'attribut } \texttt{childPos}= 2 \\ 0 & \text{sinon} \end{cases}$$

$$f_k^{(2)}(y_n, y_{ni}, \mathbf{x}, n) = \begin{cases} 1 & \text{si } y_n = 0,\ y_{ni} = \texttt{title} \\ & \quad \text{et } x_n \text{ a l'attribut } \texttt{nbChildren}= 2 \\ 0 & \text{sinon} \end{cases}$$

Une fois les fonctions de caractéristiques générées pour toutes les cliques de tous les couples de l'échantillon d'apprentissage, l'algorithme calcule pour chacune de ces fonctions $f_k \in \mathcal{F}$ sa fréquence $\omega(f_k, S)$ dans l'échantillon d'apprentissage S. Cette mesure est le rapport entre le nombre de fois parmi tous les exemples de S où la fonction de caractéristiques f_k retourne vrai et le nombre de fois où le test sur l'annotation que contient f_k est satisfait. Ainsi, cette fréquence est maximale quand, à chaque fois que le test sur l'observation est satisfait, la fonction retourne vrai. À l'inverse, si le test sur l'observation n'a que peu d'incidence sur la valeur de la fonction de caractéristiques, la fréquence associée est faible. Ainsi, $\omega(f_k, S)$ permet de mesurer l'apport du test sur l'observation que contient la fonction de caractéristiques f_k. Les fonctions de caractéristiques dont la fréquence est inférieure au seuil σ passé en paramètre de la procédure de génération sont ensuite éliminées afin de ne conserver que les fonctions considérées comme pertinentes. Ainsi, toutes les fonctions de caractéristiques structurelles sont conservées, leur fréquence ayant toujours la valeur 1.

Enfin, l'algorithme 8 retourne l'ensemble $\mathcal{F}$. Cet ensemble de fonctions de caractéristiques est délibérément indépendant du domaine d'application considéré.

Algorithme 8 Algorithme de génération de fonctions de caractéristiques.

Entrée: Un échantillon d'apprentissage S de couple $\{(\mathbf{x}^j, \mathbf{y}^j)\}_{j=1}^{j=m}$, un seuil σ de fréquence
1: Calcul des attributs de pré-traitement
2: $\mathcal{F} = \emptyset$
3: **pour chaque** $(\mathbf{x}^j, \mathbf{y}^j) \in S$ **do**
4: **pour chaque** $c \in \mathcal{C}(\mathbf{y}^j)$ *# pour toutes les cliques de* y^j **do**
5: Ajouter à l'ensemble $\mathcal{F}$ la fonction de structure correspondant à y_c^j
6: **pour chaque** test t sur l'observation possible **do**
7: $res = eval(t, \mathbf{x}^j, c)$
8: $f = genFct(y_c^j, t, res)$ *# générer la fonction qui teste les labels* y_c *et qui vérifie que le test* t *prend la valeur* res
9: Ajouter f à $\mathcal{F}$
10: **fin pour**
11: **fin pour**
12: **fin pour**
13: Calculer les fréquences de chaque fonction de $\mathcal{F}$
14: Supprimer les fonctions dont la fréquence est inférieure à σ
Sortie: $\mathcal{F}$

Cet algorithme est aussi paramétrable selon le modèle de dépendances considéré. Ainsi, si on utilise les 1-CRFs, seules les fonctions de caractéristiques sur les cliques à 1 nœuds avec un test sur l'observation sont générées. Dans le cas des 2-CRFs, les fonctions de caractéristiques sur les cliques "père-fils" sont ajoutées. Celles-ci peuvent ou non contenir un test sur l'observation. Enfin, pour les 3-CRFs, les fonctions de caractéristiques sur les

cliques triangulaires sont aussi ajoutées, et elles peuvent aussi contenir un test portant sur l'observation.

A.3 Limitations

S'il est très utile dans le cadre de nos expériences et de la composition de champs aléatoires conditionnels, cet algorithme est toutefois largement perfectible. En effet, la génération de fonctions de caractéristiques véritablement discriminantes passe souvent par la présence de plusieurs tests sur l'observation. Avec notre algorithme, la multiplication de ces tests est possible. Toutefois, il faudrait utiliser une méthode d'évaluation et de sélection des fonctions de caractéristiques plus avancée que celle que nous avons proposée afin d'éviter une forte augmentation du nombre de fonctions dans l'ensemble $\mathcal{F}$. Dans ce domaine, [McCallum, 2003] propose une algorithme d'induction de fonctions de caractéristiques pour les champs aléatoires conditionnels. Le principe de cet algorithme est de construire itérativement l'ensemble des fonctions de caractéristiques en réalisant des conjonctions de ces fonctions et en ne conservant que celles qui, lorsqu'elles sont utilisées, augmentent la log-vraisemblance conditionnelle. Cet algorithme de génération de fonctions de caractéristiques se base sur les travaux de [Pietra et al., 1997], traitant de la génération de fonctions de caractéristiques dans des champs aléatoires modélisant une probabilité jointe.

Bibliographie

[Abeillé et al., 2003] Abeillé, A., Clément, L., and Toussenel, F. (2003). *Treebanks : Building and Using Parsed Corpora*, chapter Building a Treebank for French, pages 165–188. Kluwer, abeillé ed. edition.

[Awasthi et al., 2007] Awasthi, P., Gagrani, A., and Ravindran, B. (2007). Image modeling using tree structured conditional random fields. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 2060–2065.

[Berger, 1999] Berger, A. (1999). Error-correcting output coding for text classification. In *IJCAI'99 : Workshop on machine learning for information filtering*, Stockholm, Sweden.

[Bex et al., 2004] Bex, G. J., Neven, F., and den Bussche, J. V. (2004). Dtds versus xml schema : A practical study. In *Proceedings of the Seventh International Workshop on the Web and Databases, WebDB 2004*, pages 79–84, Maison de la Chimie, Paris, France.

[Bex et al., 2006] Bex, G. J., Neven, F., Schwentick, T., and Tuyls, K. (2006). Inference of concise dtds from xml data. In *in Proceedings of 32nd Conference on Very Large databases - VLDB*, pages 115–126.

[Brill, 1993] Brill, E. D. (1993). *A corpus-based approach to language learning*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA.

[Byrd et al., 1995] Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Computing*, 16(5) :1190–1208.

[Carme, 2005] Carme, J. (2005). *Inférence de requêtes dans les arbres et applications à l'extraction d'informations sur le Web*. PhD thesis, Université Charles-de-Gaulle - Lille 3.

[Carme et al., 2007] Carme, J., Gilleron, R., Lemay, A., and Niehren, J. (2007). Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1) :33–67.

[Carme et al., 2004a] Carme, J., Lemay, A., and Niehren, J. (2004a). Learning node selecting tree transducer from completely annotated examples. In *7th International Colloquium on Grammatical Inference*, volume 3264 of *Lecture Notes in Artificial Intelligence*, pages 91–102. Springer Verlag.

[Carme et al., 2004b] Carme, J., Niehren, J., and Tommasi, M. (2004b). Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 105 – 118. Springer Verlag.

[Carreras and Marquez, 2005] Carreras, X. and Marquez, L. (2005). Introduction to the CoNLL-2005 Shared Task : Semantic Role Labeling. In *Proceedings of CoNLL-2005.*

[Chidlovskii and Fuselier, 2005] Chidlovskii, B. and Fuselier, J. (2005). A probabilistic learning method for xml annotation of documents. In *Proceedings IJCAI, 19th International Joint Conference on Artificial Intelligence.*

[Cohen et al., 2003] Cohen, W., Hurst, M., and Jensen, L. (2003). *Web Document Analysis : Challenges and Opportunities*, chapter A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific.

[Cohn and Blunsom, 2005] Cohn, T. and Blunsom, P. (2005). Semantic role labelling with tree conditional random fields. In *CoNLL'05 : Proceedings of The Ninth Conference on Natural Language Learning.*

[Cohn et al., 2005] Cohn, T., Smith, A., and Osborne, M. (2005). Scaling conditional random fields using error-correcting codes. In *ACL '05 : Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 10–17, Morristown, NJ, USA. Association for Computational Linguistics.

[Crammer and Singer, 2002] Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2 :265–292.

[Curran and Wong, 1999] Curran, J. R. and Wong, R. K. (1999). Transformation-Based Learning for Automatic Translation from HTML to XML. In *Proceedings of the Fourth Australasian Document Computing Symposium (ADCS99).*

[Daumé III et al., 2006] Daumé III, H., Langford, J., and Marcu, D. (2006). Search-based structured prediction.

[Daumé III and Marcu, 2005] Daumé III, H. and Marcu, D. (2005). Learning as search optimization : Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany.

[Denoyer and Gallinari, 2006] Denoyer, L. and Gallinari, P. (2006). Report on the xml mining track at inex 2005 and inex 2006. In *proceedings of INEX 2006.*

[Dhamankar et al., 2004] Dhamankar, R., Lee, Y., Doan, A., Halevy, A., and Domingos, P. (2004). imap : discovering complex semantic matches between database schemas. In *SIGMOD '04 : Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 383–394, New York, NY, USA. ACM Press.

[Doan, 2002] Doan, A. (2002). *Learning to Map between Structured Representations of Data.* PhD thesis, Univ. of Washington-Seattle. Received the ACM Doctoral Dissertation Award in 2003.

[Doan et al., 2001] Doan, A., Domingos, P., and Halevy, A. (2001). Reconciling schemas of disparate data sources : A machine-learning approach. In *Proceedings of the ACM SIGMOD Conference*, pages 509–520.

[Doan et al., 2003] Doan, A., Domingos, P., and Halevy, A. Y. (2003). Learning to match the schemas of data sources : A multistrategy approach. 50(3) :279–301.

[Forney, 1973] Forney, G. D. (1973). The viterbi algorithm. *Proceedings of The IEEE*, 61(3) :268–278.

[Freitag and McCallum, 2000] Freitag, D. and McCallum, A. (2000). Information extraction with hmm structures learned by stochastic optimization. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 584–589. AAAI Press / The MIT Press.

[Freitag and McCallum, 1999] Freitag, D. and McCallum, A. K. (1999). Information extraction with hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction.*

[Gallinari et al., 2005] Gallinari, P., Wisniewski, G., Denoyer, L., and Maes, F. (2005). Stochastic models for document restructuration. In *Proceedings of ECML Workshop On Relational Machine Learning.*

[Gilleron et al., 2006] Gilleron, R., Marty, P., Tommasi, M., and Torre, F. (2006). Interactive tuples extraction from semi-structured data. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence*, volume P2747, pages 997–1004. IEEE Comp. Soc. Press.

[Hammersley and Clifford, 1971] Hammersley, J. M. and Clifford, P. (1971). Markov fields on finite graphs and lattices.

[Hsu and Lin, 2002] Hsu, C. and Lin, C. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2) :415–425.

[Jelinek, 1985] Jelinek, F. (1985). Markov source modeling of text generation. *The Impact of Processing Techniques on Communications*, (91).

[Jordan et al., 1999] Jordan, M. I., Ghahramani, Z., Jaakkola, T., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37(2) :183–233.

[Kashima and Tsuboi, 2004] Kashima, H. and Tsuboi, Y. (2004). Kernel-based discriminative learning algorithms for labeling sequences, trees, and graphs. In *ICML '04 : Proceedings of the twenty-first international conference on Machine learning*, page 58, New York, NY, USA. ACM Press.

[Kohavi, 1995] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1137–1145. Morgan Kaufmann.

[Kosala et al., 2003] Kosala, R., Bruynooghe, M., Van den Bussche, J., and Blockeel, H. (2003). Information extraction from web documents based on local unranked tree automaton inference. In *18th International Joint Conference on Artificial Intelligence*, pages 403–408. Morgan Kaufmann.

[Kristjansson et al., 2004] Kristjansson, T. T., Culotta, A., Viola, P., and McCallum, A. (2004). Interactive information extraction with constrained conditional random fields. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 2004).*

[Kurgan et al., 2002] Kurgan, L., Swiercz, W., and Cios, K. (2002). Semantic mapping of xml tags using inductive machine learning. In *Proceedings of the 2002 International*

Conference on Machine Learning and Applications (ICMLA), pages 99–109. CSREA Press.

[Lafferty et al., 2001] Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields : Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 282–289.

[Lauritzen, 1996] Lauritzen, S. L. (1996). *Graphical Models*. Oxford University Press.

[Lauritzen and Spiegelhalter, 1990] Lauritzen, S. L. and Spiegelhalter, D. J. (1990). Local computations with probabilities on graphical structures and their application to expert systems. *Readings in Uncertain Reasoning*, pages 415–448.

[Lee et al., 2000] Lee, S.-Z., ichi Tsujii, J., and Rim, H.-C. (2000). Lexicalized hidden markov models for part-of-speech tagging. In *Proceedings of 18th International Conference on Computational Linguistics*, Saarbrucken, Germany.

[Lemay et al., 2006] Lemay, A., Niehren, J., and Gilleron, R. (2006). Learning n-ary node selecting tree transducers from completely annotated examples. In *International Colloquium on Grammatical Inference*, volume 4201 of *Lecture Notes in Artificial Intelligence*, pages 253–267. Springer Verlag.

[Maes et al., 2007] Maes, F., Denoyer, L., and Gallinari, P. (2007). Xml structure mapping application to the pascal/inex 2006 xml document mining track. In Fuhr, N., Lalmas, M., Malik, S., and Kazai, G., editors, *Advances in XML Information Retrieval and Evaluation : Fifth Workshop of the INitiative for the Evaluation of XML Retrieval (INEX'06)*, Dagstuhl, Germany. Springer.

[Manning and Schütze, 1999] Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge.

[Marcus et al., 1993] Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english : the penn treebank. *Computational Linguistics*, 19(2) :313–330.

[McCallum, 2003] McCallum, A. (2003). Efficiently inducing features of conditional random fields. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 403–41, San Francisco, CA.

[McCallum et al., 2000] McCallum, A., Freitag, D., and Pereira, F. (2000). Maximum entropy Markov models for information extraction and segmentation. In *Proc. 17th International Conf. on Machine Learning*, pages 591–598.

[Murphy et al., 1999] Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference : An empirical study. In *Proceedings of Uncertainty in AI*, pages 467–475.

[Muslea et al., 2001] Muslea, I., Minton, S., and Knoblock, C. A. (2001). Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2) :93–114.

[Neal, 1993] Neal, R. M. (1993). Probabilistic inference using markov chain monte carlo methods. Technical report, Department of Computer Science, University of Toronto.

[Neven, 2002] Neven, F. (2002). Automata theory for xml researchers. *SIGMOD Rec.*, 31(3) :39–46.

[Nguyen and Guo, 2007] Nguyen, N. and Guo, Y. (2007). Comparisons of sequence labeling algorithms and extensions. In *ICML '07 : Proceedings of the 24th international conference on Machine learning*, pages 681–688, New York, NY, USA. ACM Press.

[Pietra et al., 1997] Pietra, S. D., Pietra, V. D., and Lafferty, J. (1997). Inducing features of random fields. *"IEEE Transactions on Pattern Analysis and Machine Intell."*, 19(4) :380–393.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

[Quinlan, 2004] Quinlan, R. (2004). Data mining tools see5 and c5.0. http ://www.rulequest.com/see5-info.html.

[Rabiner and Juang, 1986] Rabiner, L. R. and Juang, B. H. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1) :4–16.

[Sang and Meulder, 2003] Sang, E. F. T. K. and Meulder, F. D. (2003). Introduction to the conll-2003 shared task : language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 142–147, Morristown, NJ, USA. Association for Computational Linguistics.

[Seymore et al., 1999] Seymore, K., McCallum, A., and Rosenfeld, R. (1999). Learning hidden Markov model structure for information extraction. In *AAAI 99 Workshop on Machine Learning for Information Extraction*.

[Sjölander, 2003] Sjölander, K. (2003). An hmm-based system for automatic segmentation and alignment of speech. In *Proceedings of Fonetik*, pages 93–96.

[Skounakis et al., 2003] Skounakis, M., Craven, M., and Ray, S. (2003). Hierarchical hidden markov models for information extraction. In *IJCAI*, pages 427–433.

[Sutton and McCallum, 2005] Sutton, C. and McCallum, A. (2005). Composition of conditional random fields for transfer learning. In *HLT/EMLNLP*.

[Sutton and McCallum, 2006] Sutton, C. and McCallum, A. (2006). *Introduction to Statistical Relational Learning*, chapter An Introduction to Conditional Random Fields for Relational Learning. MIT Press, lise getoor and ben taskar edition.

[Tang et al., 2006] Tang, J., Hong, M., Li, J., and Liang, B. (2006). Tree-structured conditional random fields for semantic annotation. In *The Semantic Web - ISWC 2006*, volume 4273, pages 640–653.

[Taskar et al., 2003] Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin markov networks. In Thrun, S., Saul, L., and Sch"olkopf, B., editors, *Advances in Neural Information Processing Systems 16*. Cambridge, MA.

[Toutanova et al., 2005] Toutanova, K., Haghighi, A., and Manning, C. D. (2005). Joint learning improves semantic role labeling. In *ACL '05 : Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 589–596, Morristown, NJ, USA. Association for Computational Linguistics.

[Tsochantaridis et al., 2005] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6 :1453–1484.

[Vlist, 2002] Vlist, E. V. D. (2002). *XML Schema*. O'Reilly Media, Inc.

[Vlist, 2003] Vlist, E. V. D. (2003). *Relax NG*. O'Reilly Media, Inc.

[Wallach, 2002] Wallach, H. (2002). Efficient training of conditional random fields. Master's thesis, University of Edinburgh.

[Wallach, 2003] Wallach, H. M. (2003). Efficient training of conditional random fields. In *Proceedings of the 6th Annual CLUK Research Colloquium*, Edinburgh, U.K.

[Wen, 1991] Wen, W. X. (1991). Optimal decomposition of belief networks. In *UAI '90 : Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 209–224, New York, NY, USA. Elsevier Science Inc.

[Wetherell, 1980] Wetherell, C. S. (1980). Probabilistic languages : A review and some open questions. *ACM Comput. Surv.*, 12(4) :361–379.

Index